VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Interaktívni hrací plocha s robotickou rukou

Interactive Play Board with Robotic Arm

2019 Adam Zahatlan

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Adam Zahatlan**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Interaktivní hrací plocha s robotickou rukou
Interactive Play Board with Robotic Arm

Jazyk vypracování: čeština

Zásady pro vypracování:

Rozšiřte stávající program pro ovládání robotické ruky o dotykovou interaktivní hrací plochu (dotykové LCD).
Aplikace umožní vytvořit vlastní tvar hracího pole a robotická ruka bude manipulovat až se dvěma hracími figurkami. Pro realizaci se předpokládá využití knihovny Qt, OpenCV a OS Linux.

1. Seznamte se bakalářskou a diplomovou prací, v níž byla využita robotická ruka.
2. Seznamte se s minipočítačem UDOO Quad a možnostmi této platformy.
3. Navrhněte rozšíření stávající aplikace o řízení interaktivní hrací plochou. Navrhněte funkcionalitu, rozložení grafických prvků a zohledněte případná funkční omezení.
4. Navrhněte řídící program pro modul mikropočítače instalovaný na UDOO pro ovládání robotické ruky. Pro ovládání využijte stávající komunikační protokol.
5. Navržená řešení pro Qt aplikaci a pro mikropočítač realizujte.
6. Otestujte navržené programové řešení a dle možností jej prezentujte.

Seznam doporučené odborné literatury:

[1] Aleš Prchal, Řízení robotické ruky pomocí vícejádrového hybridního procesoru, diplomová práce, VŠB-TUO FEI katedra informatiky, 2017
[2] Jan Šrámek, Řízení robotické ruky pomocí Freescale i.MX6SX, bakalářská práce, VŠB-TUO FEI katedra informatiky, 2017
[3] Minipočítač UDOO Quad: https://www.udoo.org/udoo-dual-and-quad/
[4] Grafická knihovna Qt: https://www.qt.io/developers/
[5] Grafická knihovna OpenCV: https://www.opencv.org
[6] Ukázka aplikace: https://youtu.be/S_EnJFpY_G0

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Petr Olivka, Ph.D.**

Datum zadání: 01.09.2018

Datum odevzdání: 30.04.2019

doc. Ing. Jan Platoš, Ph.D.
vedoucí katedry

prof. Ing. Pavel Brandštetter, CSc.
děkan fakulty

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 30. dubna 2019 .....

Týmto by som chcel poďakovať Ing. Petru Olivkovi, Ph.D, ktorý mi ochotne pomáhal pri riešení problémov, ako aj za cenné rady pri vedení tejto bakalárskej práce.

Abstrakt

Táto práca sa zaoberá implementáciou rozšírení navrhnutých nad systémom riadenia robotickej ruky. Súčasne opisuje implementáciu nového riešenia hry v podobe dynamickej hracej plochy kreslenej užívateľom. Pôvodné riešenie vyžívajúce UDOO NEO je nahradené platformou UDOO QUAD, ku ktorej je pripojený 15,6" LCD dotykový panel. Ten súčasne nahrádza starú hraciu plochu pôvodne vytlačenú na papieri. Uvedené riešenie sa skladá z dvoch hlavných častí. Prvá je UDOObuntu linuxová distribúcia bežiaca na procesore i.MX6Quad, zobrazujúca grafické užívateľské rozhranie aplikácie RoboticArm. Zaisťuje komunikáciu s užívateľom a predávanie vypočítaných dát o pohybe robotickej ruky mikrokontroléru SAM3X8E dosky Arduino DUE. Druhá časť beží na samotnom miktrokontroléri SAM3X8E, ktorý obstaráva PWM úlohy pre riadenie servomotorov robotickej ruky. Tie sú založené na real-time operačnom systéme FreeRTOS. Výmena dát je realizovaná pomocou UART sériovej linky.

Klíčová slova: UDOO, QUAD, i.MX6Quad, Arduino, DUE, SAM3X8E, FreeRTOS, Atmel, Studio, Qt, Creator

Abstract

This thesis is focused on implementation of changes over a robotic arm control system. Simultaneously it describes implementation of a new game form of playboard dynamically drawn by an user. The original solution, which used UDOO NEO is replaced by UDOO QUAD with connected 15,6" LCD touch panel. It also replaces old version of a track which was printed on a sheet of paper. This solution consists of two main parts. The first is linux distribution UDOObuntu running on a i.MX6Quad processor, showing graphical interface of a RoboticArm application. It provides communication with the user and passes calculated data of movement of the robotic arm to a SAM3X8E microcontroller. Second part is running on the SAM3X8E microcontroller itself, which performs PWM tasks for controlling the robotic arm servos. These are based on a real-time operating system FreeRTOS. Data exchange is realized over a UART serial line.

Key Words: UDOO, QUAD, i.MX6Quad, Arduino, DUE, SAM3X8E, FreeRTOS, Atmel, Studio, Qt, Creator

Obsah

Seznam použitých zkratek a symbolů **9**

Seznam obrázků **10**

Seznam tabulek **11**

Seznam výpisů zdrojového kódu **12**

1 Úvod **13**

2 Zoznámenie sa s pôvodným stavom **14**
2.1 Stávajúce hardwarové riešenie 14
2.2 Opis stávajúceho hardwarového riešenia 15
2.3 Ovládanie servomotora 17
2.4 Aplikácia RoboticArm 17

3 Návrh nového riešenia **20**
3.1 Vývojová doska UDOO QUAD 21
3.2 Architektúra ARM 23
3.2.1 Procesor i.MX6Quad 24
3.2.2 Mikrokontroléry série SAM3X/A, mikrokontrolér SAM3X8E 24
3.3 Rozširujúca doska pre Arduino DUE 27
3.4 Linuxová distribúcia UDOObuntu 2 28
3.4.1 Prvé spustenie systému UDOObuntu 30
3.5 Programové riešenie pre robotickú ruku 31
3.5.1 Medzi procesorová komunikácia 31
3.5.2 Arduino IDE/Atmel Studio IDP 33
3.5.3 FreeRTOS 34
3.5.4 Nahratie programu na SAM3X8E mikrokontrolér 34
3.5.5 Návrh riadiaceho programu 35
3.5.6 Sériová komunikácia 36
3.6 Aplikačné riešenie pre robotickú ruku 37
3.6.1 Prenos aplikácie RoboticArm 37
3.6.2 Úpravy aplikácie RoboticArm 39
3.6.3 Matematické výpočty aplikácie RoboticArm 40
3.6.4 Vlastné výpočty pre kreslenú dráhu 42

4 Test modifikovaného riešenia **46**
4.1 Test medzi procesorovej komunikácie 46
4.2 Test medzi procesorovej kominukácie z externého počítača 47
4.3 Test externého programátora pre Atmel Studio 48
4.4 Test generovania šírky pulzu pomocou PWM 50
4.5 Test riadiaceho programu 50
4.6 Test generovania dráhy 52
4.7 Test vypočítaných hodnôt pre umiestňovanie figúrky na dráhe 53

5 Závěr **55**

Literatura **56**

Přílohy **57**

A Konfigurácia Atmel Studia, pridanie vlastného programátora **58**

B Rozširujúca doska pre Arduino DUE **60**

C Inštalácia QtCreator-u na architektúre ARM, nastavenie aplikácie RoboticArm **61**

D Obsah priloženého média **63**

Seznam použitých zkratek a symbolů

ADC/DAC	–	Analog to Digital / Digital to Analog converter
ARM	–	Advanced RISC Machine
ARMHF	–	ARM Hard floating point support
BOD	–	Brown-out Detector
CAN	–	Controller Area Network
DVBT	–	Televízne digitálne pozemské vysielanie
DYI	–	Do-It-Yourself
FreeRTOS	–	Free real time operation system
GFLOPS	–	GigaFLOPS, skratka pre počet operácií s pohyblivou desatinnou čiarkou za sekundu
GND	–	Uzemnenie
GPIO	–	General Purpose Input/Output
IDE/IDP	–	Integrated Development Environment / Platform
JTAG	–	Rozhranie pre programovanie FLASH pamätí
LTS	–	Long Tme Support
LVDS	–	Low-voltage differential signaling
MAC	–	Media Access Control address
MCI	–	Media Control Interface
MOSFET	–	Tranzistory s ovládacou elektródou oddelenou izolačnou vrstvou
MPE	–	Media Processing Engine
MPU	–	Memory Protection Unit
NAND	–	NOT-AND hradlo
POR	–	Power-on-Reset
PWM	–	Pulse Width Modulation
RTD/RTT	–	Round-Trip Delay / Round-Trip Time
SMC	–	Static Memory Controller
SPI	–	Serial Peripheral Interface
TC	–	Timer Counter
TWI	–	Komunikačná zbernica
U(S)ART	–	Universal (Synchronous) Asynchronous Receiver/Transmitter

Seznam obrázků

1 Pôvodné riešenie diplomovej práce ukazujúce svoju funkčnosť 14
2 Pôvodné blokové schéma zapojenia 15
3 Robotická ruka Lynxmotion, 1 - kruhová základňa, 2, 3, 4 - pohyblivé ramená, 5 - kliešte pre úchyt predmetov (červené), vyznačené dôležité časti H, L1, L2, L3 (zelené) 16
4 Princíp ovládania servo motora pomocou PWM 17
5 Režim ovládania servomotorov roboticej ruky aplikácie RoboticArm 19
6 Režim zjednodušenej hry „Človeče, nehnevaj sa!“ aplikácie RoboticArm 19
7 15,6“ LCD dotykový panel UDOO s kapacitnou vrstvou 20
8 Vývojová doska UDOO QUAD [5] 23
9 Blokový diagram procesoru i.MX6Quad 25
10 Blokový diagram mikrokontroléra SAM3X8E 26
11 Pinout Arduina DUE 28
12 Navrhnutá schéma rozširujúcej dosky 29
13 Program Win32DiskImager a nahrávanie obrazu na SD kartu 31
14 Blokové schéma komunikácie medzi procesorom i.MX6Quad a SAM3X8E mikrokontrolérom 32
15 Atmel Studio 7, ASF Wizard 36
16 Robotická ruka fungujúca na vývojovej doske UDOO QUAD 38
17 Nákres s výpočtami pre kreslenú dráhu 44
18 Hotový systém ukazujúci svoju funkčnosť 45
19 Modifikácia sériovej linky programu minicom 46
20 Ukážka generovania pulzu 1.5 ms (obr. 20a) v časovom intervale 20 ms (obr. 20b) na osciloskope 50
21 Dráha nakreslená užívateľom v aplikácií RoboticArm 53
22 Umiestnenie figúrky na kreslenej hracej ploche 54
23 Poloha figúrok, kedy dochádzalo k ich kolíziam 54
24 Nastavenie programmeru 59
25 Modifikované Atmel Studio 59
26 Ukážka vyrobenej a zapojenej rozširujúcej dosky 60
27 Navrhnuté DPS rozširujúcej dosky 60
28 Ukážka nastavenia programu Qt Creator 62

Seznam tabulek

1 Zoznam uchovaných kanálov z pôvodného riešenia 16
2 Vyčítané parametre triedy senddata.cpp pre prevod uhlu na kroky servomotora . 42

Seznam výpisů zdrojového kódu

1 Pseudokód pre prevod uhlov na šírku pulzov servomotorov 42
2 Konštanty triedy drawgame.cpp pre kreslenú dráhu 45
3 Výpis zo sériovej linky programu minicom pre /dev/ttymxc3 47
4 Výpis zo sériovej linky pri pripojenej k procesoru i.MX6Quad 47
5 Sekvencia nahratia programu externým programátorom 49
6 Ukážka komunikácie po UART linke 51
7 Obsah súboru DueProgrammmer.bat 58

1 Úvod

Cieľom tejto diplomovej práce je návrh riadiaceho programu pre ovládanie robotickej ruky spoločnosti Lynxmotion ako aj úprava užívateľskej aplikácie RoboticArm. Obe veci boli dodané našej škole spoločnosťou Freescale, ktorú v medzičase odkúpila spoločnosť NXP. Slúžili ako demonštrácia, čo všetko sa dá na i.MX platforme procesorov založených na architektúre ARM vyrobených spoločnosťou NXP dokázať.

Táto práca je rozdelená celkom do troch samostatných celkov, ktoré sú postupne opísané. V prvej kapitole prebehne zoznámenie so stávajúcim riešením ovládania robotickej ruky, ktoré sa už zjednodušilo v rámci inej práce. Toto zjednodušenie prinieslo väčšiu stabilitu celého systému, zníženie počtu potrebných periférií a jednoduchosť riešenia. Súčasne ostali uchované hlavné časti pôvodného riešenia. Prvou je robotická ruka Lynxmotion L6AC a druhou platforma i.MX procesorov firmy NXP. Súčasne nesmiem zabudnúť spomenúť aj tretiu hlavnú časť, a tou je aplikácia RoboticArm, ktorá sa stará o užívateľský zážitok a súčasne preberá plnú kontrolu nad pohybmi samotnej robotickej ruky.

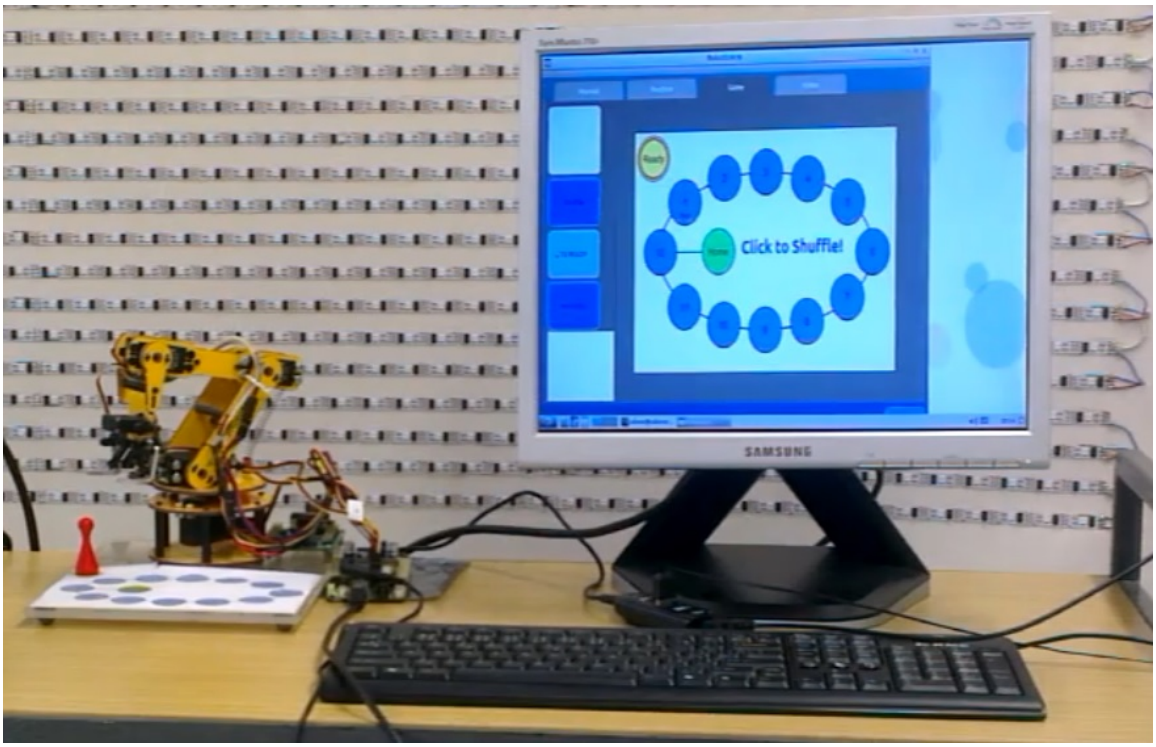
V druhej kapitole sa budem venovať opisu zmien nad riešením opísaným v prvej kapitole. Tieto riešenia budú navrhnuté s cieľom ďalej zdokonaliť pôvodné riešenie, zjednodušiť jeho funkčnosť a rozšíriť paletu ponúkaných prvkov. Najprv prebehne zoznámenie s novou platformou UDOO QUAD, ktorá bude použitá k nahradeniu pôvodnej riadiacej dosky UDOO NEO. Taktiež sa opíšu jej hlavné prednosti, možnosti a dostupný hardvér použitý ako súčasť riešenia. Ďalej bude opísaná platforma Arduino DUE, ktorá je súčasťou vývojovej dosky UDOO QUAD. Tá preberie funkcionalitu riadenia robotickej ruky. Založená bude nad real-time operačným systémom FreeRTOS a jej úlohou sa stane správne nastavenie PWM výstupov a príjem dát. V rámci softvérovej časti bude užívateľ zoznámený s linuxovou distribúciou UDOObuntu vo verzii 2, ktorá zaberie svoje miesto na procesore i.MX6Quad. Súčasne sa na nej pokúsim rozbehnúť aplikáciu RoboticArm, ktorá slúži na obsluhu robotickej ruky podľa toho, aký režim ovládania je zvolený užívateľom. Nad touto aplikáciou bude pridaná funkcionalita, ktorú taktiež v tejto kapitole opíšem.

V poslednej kapitole opíšem testy, ktoré boli vykonané v priebehu práce na projekte. Tieto testy sú zamerané na správnu funkčnosť riešenia, jeho stabilitu a prípadné problémy súvisiace s navrhnutým riešením.

Taktiež sú súčasťou práce prílohy, kde užívateľ nájde postupy, ako upraviť Atmel Studio pre programovanie vývojovej dosky Arduino DUE, postup spustenia aplikácie RoboticArm a DPS pre vyrobenie vlastnej rozširujúcej dosky.

2 Zoznámenie sa s pôvodným stavom

Toto riešenie už bolo škole v minulosti dodané spoločnosťou NXP (pôvodne freescale) ako ukážková sada pre demonštráciu možnosti výpočetnej techniky založenej na platforme *i.MX 6SoloX*. Jednalo sa o možnosť ovládania robotickej ruky užívateľom pomocou grafického užívateľského rozhrania aplikácie RoboticArm zobrazovaného na dotykovom displeji. Užívateľ mohol kontrolovať pohyb robotickej ruky ako aj jej jednotlivých serv. Užívateľ si taktiež mohol zahrať zjednodušenú verziu hry *„Človeče, nehnevaj sa!“*, ktorá bola vytlačená na predpripravenej hracej ploche.

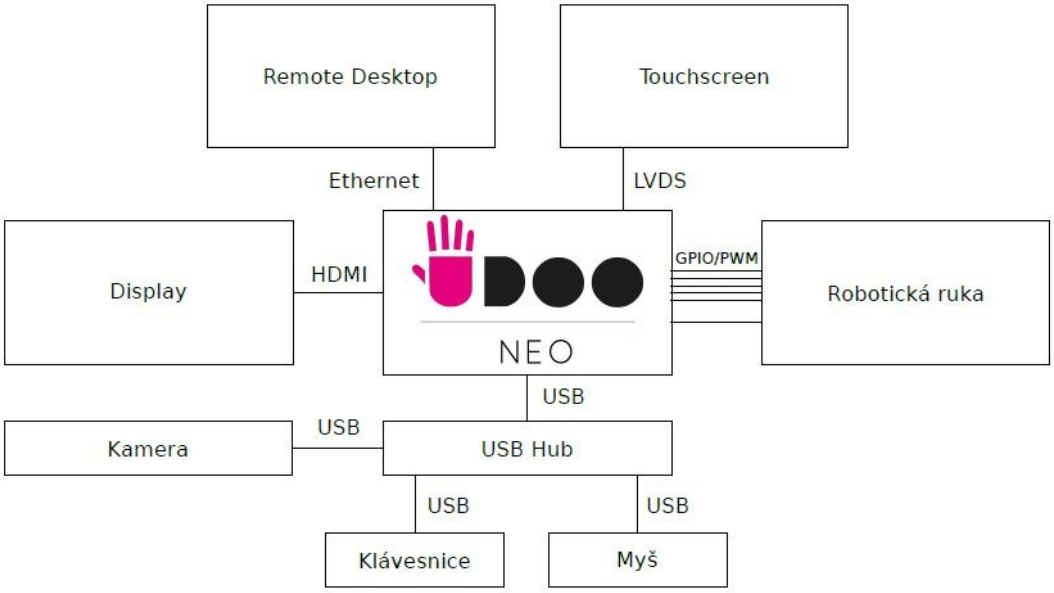


Obrázek 1: Pôvodné riešenie diplomovej práce ukazujúce svoju funkčnosť

2.1 Stávajúce hardwarové riešenie

V súčasnosti je už toto riešenie upravené Ing. Alešom Prchalom ukázané na obr. 1, ktorý ho upravoval ako diplomovú prácu [1], ktorú som počas svojej práce dostal ako odporučenú odbornú literatúru. V tomto riešení nahradil vývojovú dosku *i.MX Sabre SD* od firmy NXP osadenú čipom *i.MX 6SoloX* vývojovou doskou UDOO NEO vo variante full, ktorá taktiež nahrádza *Lynxmotion Servo Controller SSC-32* [2] pre kontrolovanie serv robotickej ruky, nakoľko vývojová doska UDOO NEO obsahuje potrebné PWM a GPIO porty spĺňajúce požiadavky generovania premenného výstupu na báze signálu meniaceho sa pomocou hodín. K tejto doske

bol pripojený LCD dotykový displej s kapacitnou vrstvou pre interakciu s používateľom, Genius webkamera, klávesnica a myš ako je znázornené na obr. 2



Obrázek 2: Pôvodné blokové schéma zapojenia

2.2 Opis stávajúceho hardwarového riešenia

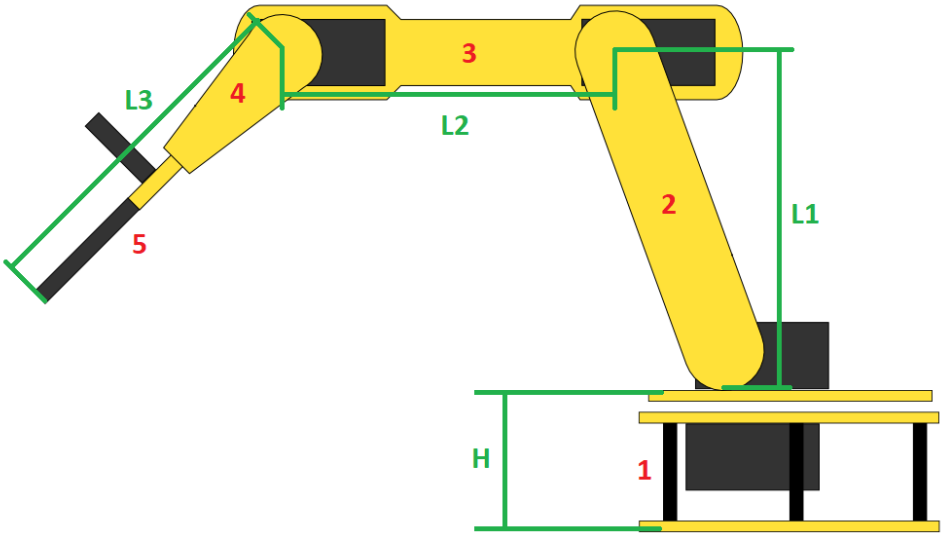
Ako som už naznačil, riešenie dodávané firmou NXP bolo upravené ako diplomová práca.

Hlavnou časťou celého riešenia je robotická ruka Lynxmotion L6AC vyrobená a dodaná firmou Lynxmotion. Táto ruka sa skladá z troch ramien rôznej dĺžky, klieští pre úchyt a kruhovej základne, ktoré pre zmenu polohy polohy používajú celkom šesť servomotorov značky HITEC, z toho ich je 5 typu *HS-422* a jedno typu *HS-85BB.*

Toto umožní ruke päť rôznych pohybov k jej plnej funkčnosti v priestore. Ten má obmedzený polomer, smer a dĺžku natiahnutia. Pre každú časť ruky zobrazenej červeno na obr. 3 (základňa, ramená, kliešte), je použitý jeden servomotor až na kĺb ramena. Kvôli obmedzenému výkonu servomotorov a váhe ruky pri maximálnej dĺžke sú v tomto kĺbe použité dve spriahnuté servá k optimalizovaniu točivého momentu a minimalizovaniu chvení ramena.

Pre kontrolovanie týchto servomotorov sú v tejto práci použité dve riešenia, pričom obe sú zhodné generovaným výstupom a teda aj funkčnosťou:

1. Prvé riešenie je realizované pomocou PWM, ktoré využíva hodín k tomu, aby sa kontroloval čas, počas ktorého sa na výstupu generuje log. 1, inak je na výstupe generovaných log. 0. Toto sa generuje v danom časovom úseku, pre správne riadenie pohybu servomotorov robotickej ruky.



Obrázek 3: Robotická ruka Lynxmotion, 1 - kruhová základňa, 2, 3, 4 - pohyblivé ramená, 5 - kliešte pre úchyt predmetov (červené), vyznačené dôležité časti H, L1, L2, L3 (zelené)

2. Druhé riešenie je realizované pomocou GPIO, ktoré využíva vlastností portov, ktoré môžu byť nastavené striedavo ako vstupné alebo výstupné a umožňuje periodické vypínanie a zapínanie jednotlivých portov. Toto sa realizuje rovnako ako u PWM v danom časovom úseku po určitú dobu.

Obe z uvedených riešení kontroluje všetky servá robotickej ruky v rovnaký čas. Kontrola jednotlivých servomotorov prebieha na rôznych výstupných pinoch dosky UDOO NEO, ktoré sú označované ako kanály podľa pôvodného riešenia firmy NXP so servo-kontrolérom *SSC-32* [2]. Celkovo je potreba 6 kanálov, pričom na kanály *1* a *17* sa privádza rovnaký signál kvôli kontrolovaniu oboch spriahnutých serv ramena súčasne, ako je vidieť v tabuľke 1.

Súčasne existuje akýsi virtuálny kanál *7*, ktorý pri dotaze naň vráti aktuálnu polohu všetkých PWM kanálov ako aj čas, za ktorý sa na toto miesto dostali, pokiaľ už bola poloha ruky inicializovaná (už bolo pohnuté niektorým ramenom aspoň v jednom smere).

Rameno	Číslo kanálu
1	0
2	1, 17
3	2
4	3
5	4
Virtuálny kanál	7

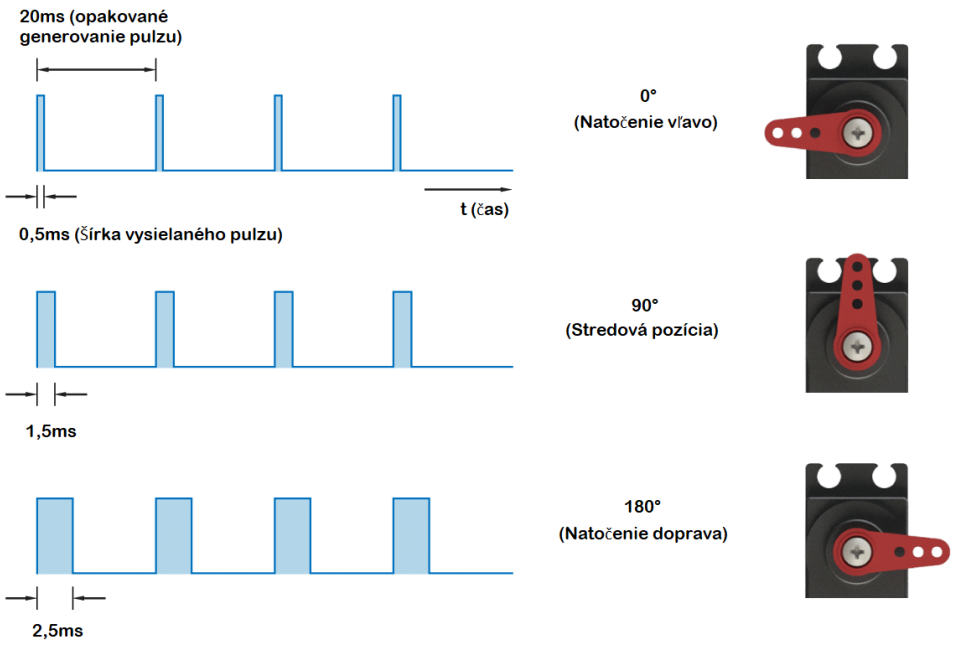
Tabulka 1: Zoznam uchovaných kanálov z pôvodného riešenia

Toto opísané riešenie umožnilo odstránenie nadbytočného hardvéru ako aj zjednodušenie kontroly robotickej ruky a celkové zvýšenie stability a výkonu systému.

2.3 Ovládanie servomotora

Servomotor je riadený pulzom privedeným na jeho signálny vstup. Celkom má servo motor 3 konektory: Signálny, +5V a GND, čo znamená, že servo je pod napätím stále, no pohybuje sa iba vtedy, keď dostane signál v podobe log. 1 na signálnom konektore.

Miera natočenia servo motora sa určuje šírkou generovaného pulzu v danom časovom úseku. Táto šírka sa u jednotlivých servo motorov líši a umožňuje im rôzne uhly natočenia. V našom prípade sú použité servá typu značky HITEC, ktorých šírka pulzu sa dá nastaviť v rozmedzí 0,5 – 2,5 ms (500 – 2500 us). Tento pulz sa kvôli správnemu pohybu ruky musí opakovať v časovom úseku 20 ms (20000 us) ako je vidieť na obr. 4.



Obrázek 4: Princíp ovládania servo motora pomocou PWM

Toto umožňuje použitým servo motorom pohyb v rozsahu cca 180° [2]. Nanešťastie nie vždy môže byť celý tento rozsah použitý v rámci rôznych nastaviteľných polôh robotickej ruky. Toto by však mohlo vyvolať problém pri náhodnom kontrolovaní ruky cez grafické rozhranie aplikácie RoboticArm, kde si môže užívateľ nastaviť sklon jednotlivých ramien robotickej ruky. RoboticArm má toto správanie korektne ošetrené a tak nemôže dôjsť k tejto chybe.

2.4 Aplikácia RoboticArm

Ako som už spomínal, ďalšou neodmysliteľnou časťou tejto práce je desktopová užívateľská aplikácia RoboticArm. Túto aplikáciu taktiež dodala firma NXP. Aplikácia je vyvinutá nad knižnicou *Qt* [3] v programe Qt Creator. Ide o celkom efektívne riešenie, aplikácia má malé odozvy a je ľahko rozšíriteľná a programovateľná, nakoľko Qt knižnica je grafickou nadstavbou

pre jazyk C++. V aplikácií sa taktiež využívajú rôzne grafické prvky ako spracovanie obrazu z kamery. K tomuto účelu sa používa knižnica *OpenCV*. Aplikácia taktiež dokáže prehrávať inštruktážne video, o čo sa stará ďalšia použitá knižnica *GStreamer*.

Aplikácia RoboticArm by sa taktiež dala rozdeliť na 2 funkčné časti. Prvou časťou je grafické užívateľské rozhranie, ktoré slúži na interakciu s užívateľom hneď v niekoľkých režimoch:

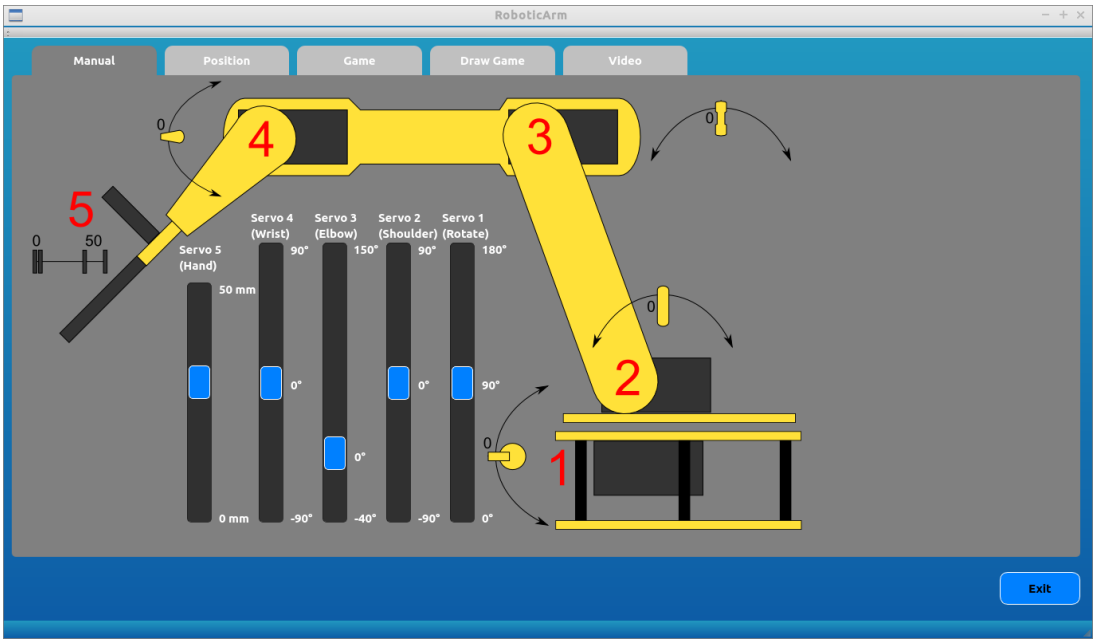
1. ***Manual*** – ide o režim, kedy užívateľ prevezme plnú kontrolu nad každým servomotorom, pričom môže plne kontrolovať jeho pohyb ako je vidieť na obr. 5. Užívateľovi je zobrazených 5 posuvníkov, pričom každý môže byť kontrolovaný iba v jednej chvíli, čo znamená že užívateľ môže hýbať iba s jedným servom súčasne.

2. ***Position*** – ďalší režim, kedy môže užívateľ do určitej miery slobodne kontrolovať pohyby robotickej ruky. Na rozdiel od prvého režimu, užívateľ môže kontrolovať celú robotickú ruku, čo znamená pohybovať s niekoľkými ramenami súčasne. Obmedzenie prichádza v nakreslenej virtuálnej hranici, ktorú užívateľ kvôli limitom dosahu robotickej ruky nemôže prekročiť. Užívateľovi sú taktiež znova k dispozícií 2 posuvníky pre kontrolovanie polomeru otočenia a rozovretia klieští robotickej ruky.

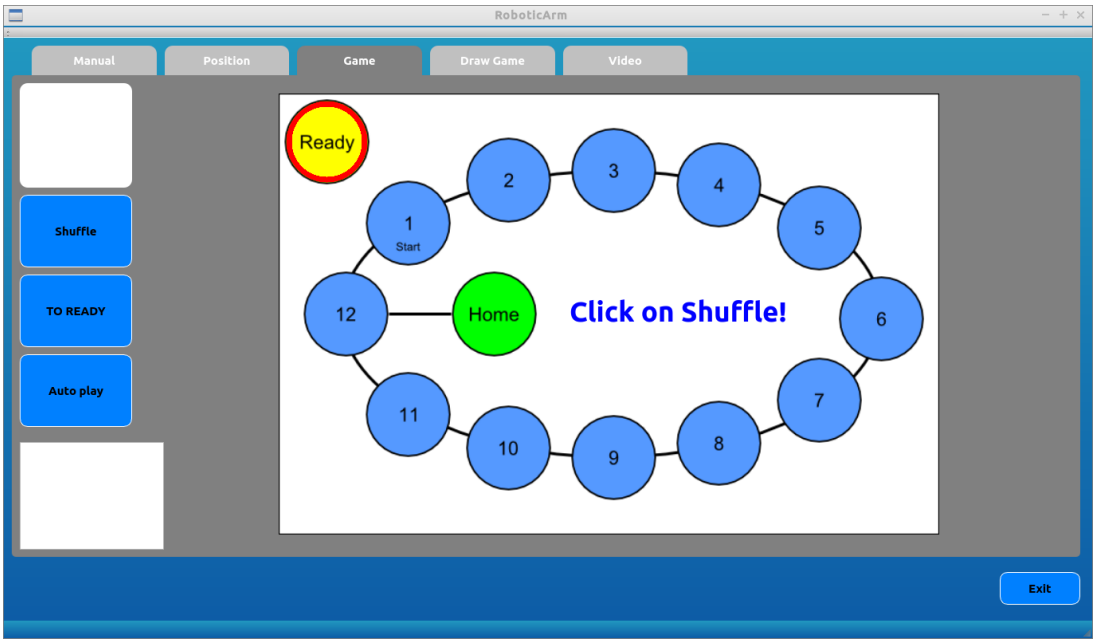
3. ***Game*** – tretí režim ponúkne užívateľovi možnosť si zahrať zjednodušenú variantu hry „Človeče, nehnevaj sa!“. Súčasná pozícia figúrky sa zobrazí užívateľovi obkreslením príslušného kruhu. Ďalej sú k dispozícií 2 okná, prvé zobrazujúce poslednú hodenú hodnotu šesťstrannej kocky a druhé aktuálny záber z kamery na miesto, kde sa s touto kockou hádže. Tento obraz sa spracováva algoritmami pre spracovanie obrazu. Ako posledné sú tu 3 tlačidlá, prvé *SHUFFLE* pre náhodne hodenie kocky bez potreby hodenia skutočnej kocky, kedy program vygeneruje náhodnú hodnotu medzi 1 a 6, druhé pre vrátenie figúrky na štartovnú pozíciu *READY* a tretie *AUTO-PLAY* pre akési automatické hranie, ktoré sa opakuje po úspešnom prejdení celej dráhy. Všetko je vidieť na obr. 6.

4. ***Video*** – posledný režim, kde si užívateľ môže prehrať demonštračné video o schopnostiach robotickej ruky ako aj aplikácie RoboticArm.

Druhá časť nie je viditeľná užívateľom a stará sa o výpočty pre robotickú ruku ako aj kontrolu, aby užívateľ nezašiel za hranice možností jej pohybu.



Obrázek 5: Režim ovládania servomotorov roboticej ruky aplikácie RoboticArm



Obrázek 6: Režim zjednodušenej hry „Človeče, nehnevaj sa!“ aplikácie RoboticArm

3 Návrh nového riešenia

Aj napriek tomu, že sa opisované riešenie môže zdať ako hotový projekt, ktorý netreba prerábať, tak stále existujú určité veci, ktoré sa dajú vylepšiť. Najviac vhodné je sa zamyslieť nad použitou dráhou. Ako bolo spomenuté, tak tá je vytlačená na kusu papiera, ktorý je prilepený na sklenenej doske hneď pod robotickou rukou. Takéto riešenie je neefektívne a neodolné voči zmenám. Užívateľ musí hrať dookola tú istú dráhu, ktorá pozostáva sumárne zo 14 bodov. To znamená, že čas jej prejdenia je krátky ako aj to, že táto dráha sa stále opakuje dookola. Jej zmena je pre obyčajného používateľa prakticky nemožná, pretože táto dráha je implementovaná v aplikácií RoboticArm. Síce sú súradnice jednotlivých bodov uložené v súbore `GamePositions`, užívateľ nemá možnosť bez bližších vedomostí o fungovaní aplikácie tento súbor upraviť. Taktiež preňho nie je možné túto dráhu upraviť v samotnej aplikácií.

Cieľom tejto práce je teda hlavne úprava hracej plochy. Úpravy budú spočívať v nahradení „dráhy“ vytlačenej na papieriku dráhou, ktorá sa bude dať za chodu programu meniť do podoby podľa toho, ako si ju užívateľ nakreslí.

Pre toto riešenie je vhodné, že spoločnosť UDOO predáva k svojím doskám originálne periférie v podobe LCD dotykových panelov, ktorých napájacie konektory sú súčasťou skoro každej vývojovej dosky UDOO. Súčasne sa odstráni ďalšia časť práce, ktorá je zbytočná. Reč je o displeji pripojenom v doske UDOO NEO, ktorý sa nahradí už spomenutým dotykovým panelom.

Pre to, aby bolo možné plne využiť potenciál kreslenej dráhy, ako aj samostatnej aplikácie, padla voľba LCD dotykového panelu s kapacitnou vrstvou na verziu o veľkosti 15,6“ s rozlíšením 1366x768 pixelov a 24 bitovou farebnou hĺbkou ukázanom na obr. 7.

Obrázek 7: 15,6“ LCD dotykový panel UDOO s kapacitnou vrstvou

Táto voľba avšak priniesla ďalší problém. Použitá vývojová doska UDOO NEO nepodporuje vybratý LCD panel. Podporuje len panel o veľkosti 7“, ktorý je žiaľ veľkostne pre aplikáciu RoboticArm aj plochu kreslenej dráhy nedostatočný.

Preto je potreba zmena vývojovej dosky. Voľba padla na dosku UDOO QUAD z rodiny DUAL/QUAD. Pre túto dosku sa rozhodlo z dôvodu vyššieho výpočtového výkonu oproti doske UDOO NEO ako aj vďaka podpore vybratého LCD panelu, ktorý sa predáva ako originálne príslušenstvo k doske UDOO QUAD.

Na ďalších stránkach teda dôjde k zoznámeniu čitateľa s podrobnými krokmi pre prenos už modifikovaného projektu na novú platformu, ako aj prerobenie aplikácie RoboticArm pre väčší užívateľský komfort a možnosť voľného kreslenia hracej dráhy.

3.1 Vývojová doska UDOO QUAD

Ako som už spomínal, pre projekt je vybraná doska UDOO QUAD z rodiny DUAL/QUAD. UDOO DUAL a QUAD sú skoro rovnaké vývojové dosky, líšia sa len počtom jadier hlavného *i.MX6Quad* procesora. Ako z názvov napovedá, UDOO DUAL má jadrá dve, zatiaľ čo QUAD má jadier štyri. Preto padla voľba na UDOO QUAD.

Platforma UDOO DUAL/QUAD bola spustená ako Kickstarterový projekt v roku 2013 ako prvá vývojová platforma spoločnosti UDOO. Záujem o túto platformu bol veľký, nakoľko kombinovala možnosti mikropočítača a mikrokontroléra v jednom. Mohla byť teda použitá aj ako vývojová doska alebo ako počítač na každodenné používanie.

UDOO QUAD je jednodoskový mikropočítač, ktorý zvládne beh ako Linuxovej distribúcie, tak aj operačného systému Android. Veľkou prednosťou tejto dosky je to, že disponuje upravenou verziou mikrokontroléra Arduino DUE, ktorý má integrovaný, čo znamená, že táto doska kombinuje dva rôzne výpočetné svety dokopy: jeden s veľkým výpočetným výkonom a druhý takzvaný DYI svet, dávajúci užívateľovi možnosť vyvíjať a kontrolovať rôzne periférie [4]. Týmto sa táto doska stáva vhodná ako pre vývoj softvéru, tak aj pre vývoj riadiacich programov. Taktiež je vhodná pre nováčikov a začínajúcich, nakoľko pre jej ovládanie a vývoj nie sú potrebné široké znalosti.

UDOO QUAD je open-hardware vývojová platforma nízkej ceny osadená procesorom architektúry *ARM i.MX6 NXP®* a ďalším mikrokontrolérom založeným na architektúre *ATMEL SAM3X8E ARM,* ktorý je súčasťou vývojovej dosky Arduino DUE.

Komponenty dosky UDOO QUAD:

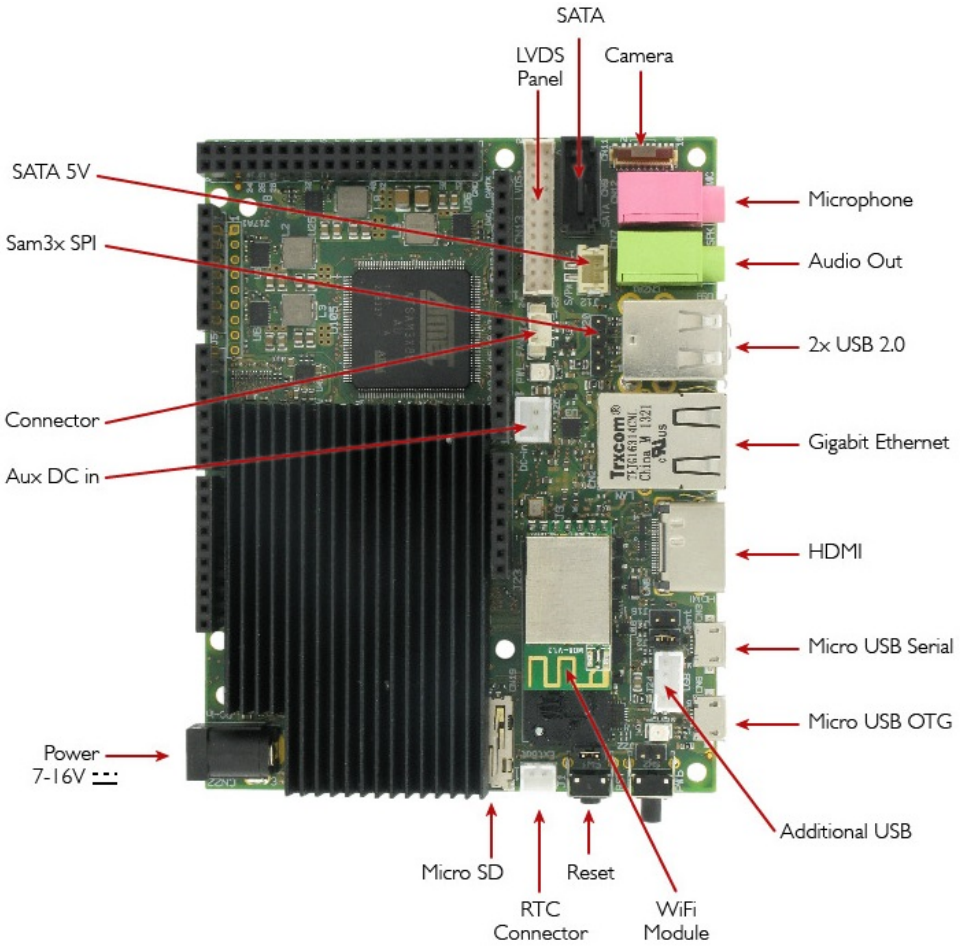
- NXP® i.MX6Quad procesor, 4 x ARM® Cortex™-A9 jadro s taktom1 GHz s inštrukčnou sadou ARMv7A,
- GPU Vivante GC 2000 pre 3D grafické výpočty + Vivante GC 355 pre 2D vektorové výpočty + Vivante GC 320 pre 2D grafické výpočty,

- Atmel SAM3X8E ARM Cortex-M3 mikrokontrolér (súčasťou upraveného Arduino DUE),
- 1GB DDR3 operačnej pamäte RAM,
- 76 plne funkčných GPIO výstupov spoločne s Arduinom DUE R3 a kombinujú rôzne typy vstupov a výstupov,
- grafické HDMI and LVDS výstupy + Touch vstup pre LVDS panel,
- 2 Micro USB (1 OTG),
- 2 USB 2.0 typu A pre pripojenie periférií a 1 USB 2.0 pinovú paticu,
- analógové Audio and Mikrofónové jacky,
- CSI pre pripojenie kamery (originálna periféria),
- Micro SD čítačka kariet pre systém (musí obsahovať boot partíciu),
- napájací konektor 6-15V DC a konektor pre externú betériu,
- Gigabit Ethernet RJ45 (10/100/1000 Mbit),
- WiFi Modul,
- SATA konektor pre pripojenie externého disku.

Ďalšou hlavnou prednosťou tejto dosky sú jej kompaktné rozmery. Doska zaberá priestor o veľkosti 11cm x 8.5cm. Hlavný procesor *i.MX6Quad* si vystačí iba s pasívnym chladením, čím sa stáva doska perfektá pre akékoľvek použitie aj v uzavretejšom priestore. Toto je zásluhou použitej architektúry ARM poskytujúcej vysoký výpočetný výkon pri nízkom prúdovom odbere.

Aby týchto predností nebolo málo, je mikrokontrolérová časť presnou kópiou Arduina DUE. To znamená, že má 3 rady pinov pre pripojenie periférií viditelné na obr. 8. Taktiež to znamená, že je táto doska kompatibilná s väčšinou shieldov pre Arduino UNO, Mega2560 a aj DUE. Samozrejme zdieľa aj obmedzenie maximálneho napätia na pinoch. Arduino DUE beží na 3.3 V, vyššie napätie privedené na vstupy alebo výstupy by mohlo dosku poškodiť.

Ako už bolo spomínané, hlavnou výpočetnou jednotkou vývojovej dosky UDOO QUAD je procesor *i.MX6Quad* rovnakej platformy ako procesor predchádzajúcej dosky UDOO NEO. Tento procesor je dodávaný spoločnosťou NXP. Založený je na architektúre *ARM* s inštrukčnou sadou *ARMv7A*. Tieto procesory sú špeciálne vyvíjané pre inovatívne multimediálne rozšírenie, čo znamená aj skratka i.MX („innovative Multimedia eXtension“).



Obrázek 8: Vývojová doska UDOO QUAD [5]

3.2 Architektúra ARM

ARM je označenie procesorov používaných vďaka svojej nízkej energetickej spotrebe hlavne v mobilných zariadeniach a rôznych vývojových doskách. Táto architektúra spôsobila v niektorých informačných systémoch revolúciu. Jej návrh sa riadil filozofiou *RISC.* Vo svojej dobe boli pokročilé ako už použitou redukovanou inštrukčnou sadou, tak použitou výrobnou technológiou, ktorá im dovolila dosiahnuť vysoké taktovacie frekvencie. Taktiež použitá 32-bitová šírka slova nebola v dobe vzniku *ARM* v roku 1984 samozrejmosťou. Spoločnosť ARM Goldings nakoniec ale časom odstúpila od výroby procesorov tejto architektúry a začala sa plne venovať ich budúcemu vývoju. Používané sú vo väčšine malej spotrebnej elektroniky, ako sú mobily, chladničky, prehrávače alebo počítačové periférie [6]. Vďaka ich dostatočnému výkonu sa postupne začínajú presadzovať v jednoduchších osobných počítačoch ako je napríklad nami použitá platforma UDOO QUAD.

Ich značný úspech stojí taktiež aj za faktom, že *ARM* procesory používajú filozofiu usporiadania všetkých periférií na jednom čipe, takzvaný *SoC – System on a Chip.* To znamená, že na

jednom čipe môžeme nájsť výpočetné jadrá, radiče pre USB, PCIe zbernicu či ethernet, ale ak grafický čip, ktorých má nami použitý procesor *i.MX6Quad* hneď niekoľko.

3.2.1 Procesor i.MX6Quad

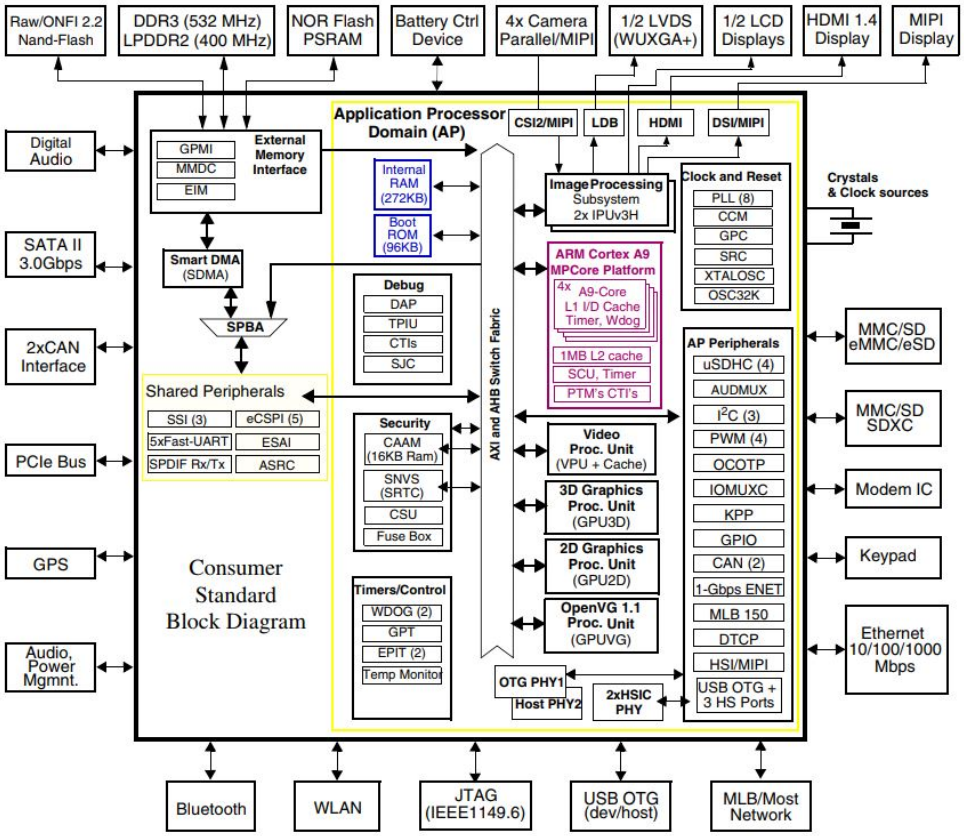
Jedná sa o druhý najvýkonnejší procesor vývojovej rady *i.MX6.* Je vyvinutý špeciálne pre kombináciu škálovateľných platforiem širokými úrovňami integrácie a energeticky účinnými procesnými možnosťami výhodnými pre multimediálne aplikácie [7].

- ***ARM®Cortex®-A9 MPCore*** **-** Srdcom procesora sú štyri jadrá ARM®Cortex®-A9 taktované a frekvenciu 1 GHz (možno nataktovať až na 1.2 GHz). Každé jadro má 32 KB L1 inštrukčnej cache a dátovej cache pamäte. Procesor disponuje 1 MB L2 cache pamäte a 32 KB inštrukčnou sadou. Výkonné výpočetné jadrá sú doplnené hneď o niekoľko grafických akcelerátorov starajúcich sa o grafické výstupy. Súčasťou procesora je privátny Timer a Watchdog a aj Cortex-A9 NEON MPE koprocesor viď na obr. 9.
- ***GPU 3D*** **-** Prvý akcelerátor je GPU Vivante GC 2000 pre 3D grafické výpočty. Výpočetný výkon je 16 – 24 GFLOPS, záleží ale na zdroji. Využíva technológie OpenGL ES 3.0 a Halti.
- ***GPU 2D (Vektorová grafika)*** – druhý akcelerátor Vivante GC 355 určený pre 2D vektorové výpočty. Beží na takte 300 – 500 MHz a je schopný vypočítať 300 – 500 M pixelov za sekundu.
- ***GPU 2D (Spracovacie zariadenie)*** – posledný akcelerátor Vivante GC 320 určený pre 2D grafické výpočty. Beží na takte 350 – 500 MHz a je schopný vypočítať 700 – 100 M pixelov za sekundu. Podporuje rozlíšenie až do 2560x1600 pixelov.
- ***Pamäť*** – procesor operuje s 1 GB DDR3 RAM 64-bitovej operačnej pamäte.
- ***SoC Memory system*** – pozostáva z niekoľkých častí: Boot ROM 96 KB, multimedia shared RAM (OCRAM, 256 KB), secure/non-secure RAM 16KB.

3.2.2 Mikrokontroléry série SAM3X/A, mikrokontrolér SAM3X8E

SAM3X/A je séria z rodiny flash mikrokontrolérov založených na vysoko výkonných 32-bitových *ARM®Cortex®-M3 RISC* procesoroch. Pracujú s rozsahom napätia od 1,62 V do 3,6 V. [8]

- ***ARM Cortex-M3 revízia 2.0*** **-** jadro procesora operujúce až na 84 MHz. Obsahuje MPU, Thumb®-2 inštrukčnú sadu, 24-bitové počítadlo SysTick a vnorený vektorový prerušovač ako je vidieť na obr. 10

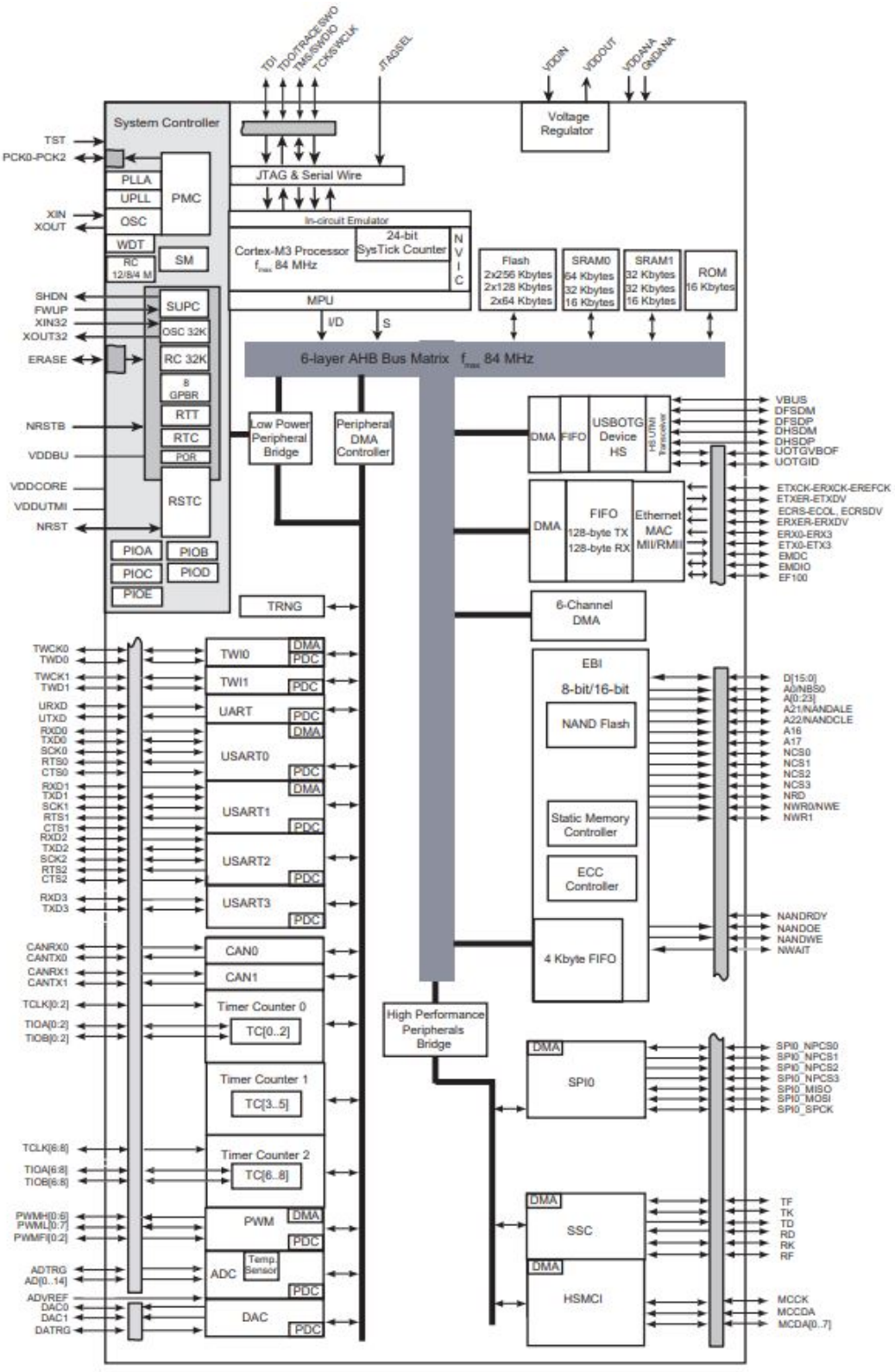


Obrázek 9: Blokový diagram procesoru i.MX6Quad

- ***Pamäte*** **-** procesory obsahujú od 256 až do 512 KB 128-bit wide access pamäte Flash, od 32 do 100 KB pamäte SRAM, 16 KB ROM pamäť pre integrované rutiny zavádzača (UART, USB) s rutinami IAP a SMC.

- ***Systém*** **-** súčasťami systému je zabudovaný regulátor napätia pre napájanie s jedným zdrojom, POR, BOD a Watchdog pre bezpečný reset, quartzové alebo keramické rezonátorové oscilátory od 3 až 20 MHz hlavné a voliteľné nízke výkony 32,768 kHz pre RTC alebo hodiny zariadenia, 8/12 MHz RC oscilátor s vysokou presnosťou a predvolenou frekvenciou 4 MHz pre rýchle spustenie zariadenia, pomalý interný RC oscilátor ako permanentné hodiny v low-power móde, 2 PLL, jedno pre hodiny a druhé externé pre vysokorýchlostné USB, teplotný senzor a až do 17 periférnych DMA (PDC) kanálov.

- ***Periférie*** **-** periférna súprava obsahuje vysokorýchlostné USB Host a Device porty s integrovaným prijímačom, Ethernet MAC, 2 CANy, vysokorýchlostné MCI pre SDIO/SD/MMC, externé bus rozhranie s NAND Flash mikrokontrolérom, 4 USART a 1 UART, 2 TWI, 4 SPI zbernice ako aj tri trojkanálové PWM časovače, 9 kanálový 32-bitový čítač časovača (TC), 8 kanálové 16-bitové PWM s 2-bitovým generátorom mŕtveho času, nízkonapäťové



Obrázek 10: Blokový diagram mikrokontroléra SAM3X8E

RTT a RTC, 256-bitové záložné registre pre všeobecné účely a 12-bitové ADC a DAC prevodníky.

Zariadenia osadené mikrokontrolérmi *SAM3X/A* majú tri režimy s nízkou spotrebou energie.

- ***SLEEP*** **-** funkcia procesoru je „uspatá“, zatiaľ čo ostatné funkcie pokračujú vo svojom behu.
- ***WAIT*** **-** v tomto móde sú všetky hodiny a časovače zastavené, ale niektoré periférie môžu byť nakonfigurované, aby „prebrali“ systém za určitých podmienok.
- ***BACKUP*** **-** v záložnom móde fungujú iba RTC, RTT a wake-up logiky.

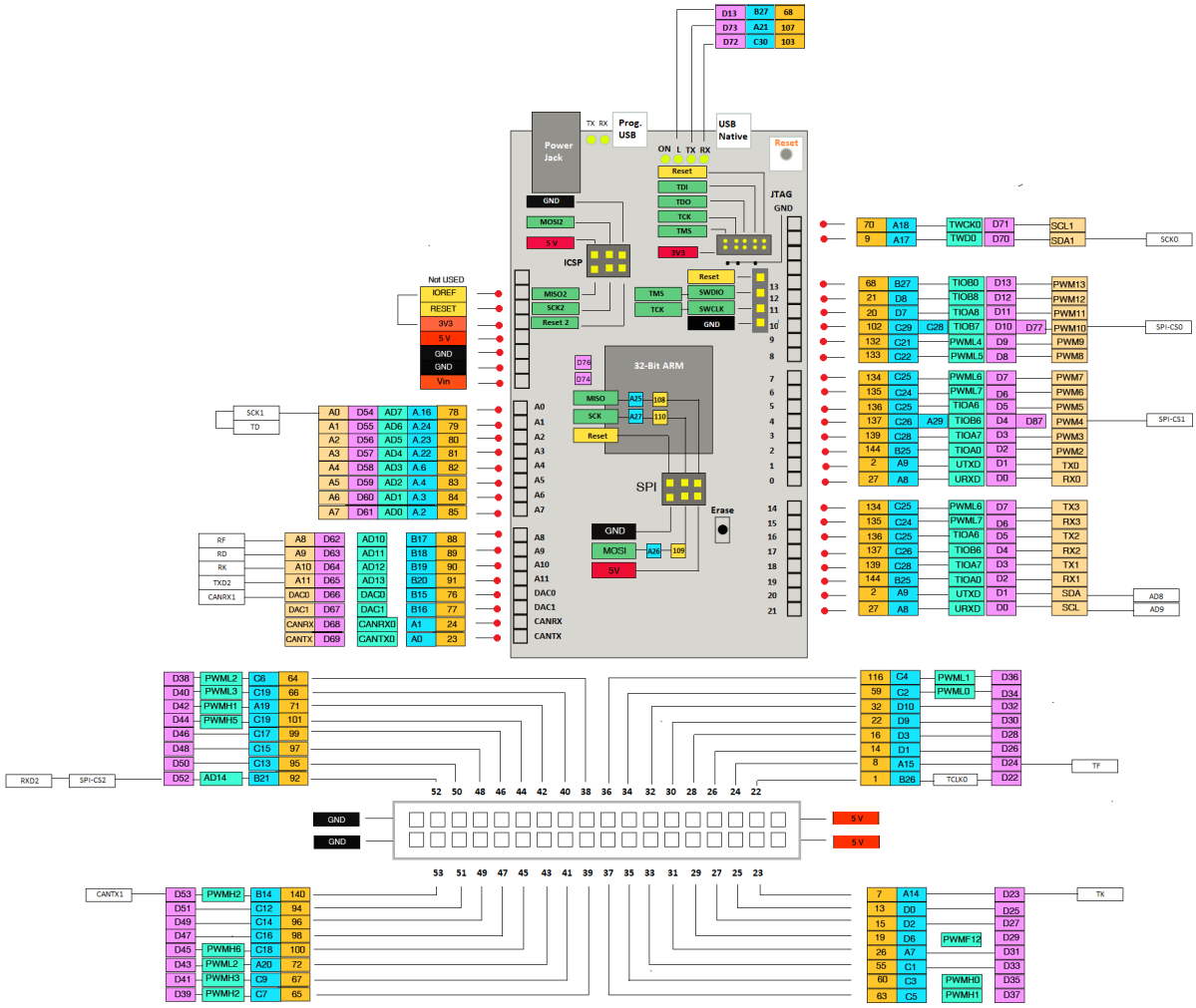
Súčasne je táto architektúra navrhnutá pre udržanie vysokorýchlostných dátových prenosov. Obsahuje viacvrstvovú zbernicovú maticu, ako aj viaceré SRAM banky, PDC a DMA kanály, ktoré umožňujú paralelne spúšťať úlohy a maximalizovať priepustnosť dát.

Každý mikrokontrolér z tejto rodiny sa odlišuje počtom periférií, zapuzdrením, ako aj veľkosťou pamäte a zbernicami navyše. Mikrokontrolér *SAM3X8E* použitý u Arduina DUE je akýmsi stropom v tejto sérií procesorov, takže obsahuje všetko a má najväčšiu možnú vnútornú pamäť.

K dispozícií je 54 digitálnych výstupov/výstupov, z ktorých 8 môže byť použitých ako PWM, 12 analógových vstupov, 4 UART, 2 DAC, 2 TWI porty a SPI a JTAG rozhranie Arduina DUE. Niektoré zo spomenutých konektorov zdieľajú spoločný pin a tak nie je možno použiť súčasne. Možno sa o nich informovať v oficiálnej dokumentácií [8]. Mňa budú zaujímať len PWM porty. Týchto 8 portov je označených v oficiálnej dokumentácií ako *PWML0 – PWML7.* V dokumentácií sa taktiež možno dočítať o portoch *PWMH0 – PWMH7.* Sú to invertované verzie portov *PWML* slúžiace pre ovládanie invertovaného obvodu. Preto sa budem výhradne zameriavať na porty *PWML.* Konkrétne sa pre ovládanie robotickej ruky používajú porty *PWML3 – PMWL7,* pretože ich piny rozmiestnené na doske sú k sebe najbližšie ako je vidieť na obr. 11.

3.3 Rozširujúca doska pre Arduino DUE

Keď som si zistil presné pozície pinov pre PWM porty z obr. 11, bolo jednoduché napojiť skúšobné servomotory, pretože Arduino DUE disponuje dvoma +5 V a GND portami. To však nestačilo pre robotickú ruku Lynxmotion L6AC, nakoľko disponuje až šiestimi servomotormi. Prvotná myšlienka použiť pre vytvorenie rozširujúcej dosky maketu shieldu pre Arduino UNO avšak nebola realizovaná, pretože nedisponovala dostatočným rozpätím a počtom pinov. Preto bola mnou a vedúcim práce navrhnutá a vyrobená rozširujúca doska pre jednoduché pripojenie a ovládanie jednotlivých servomotorov. Pre jej tvorbu bola použitá vytvorená knižnica [15] vytvorená Arduino komunitou a následne upravená mnou, pretože klon Arduina DUE zakomponovaný v doske UDOO QUAD nedisponuje všetkými pinmi ako originálna doska. DPS tejto dosky je možno nájsť v prílohe B.

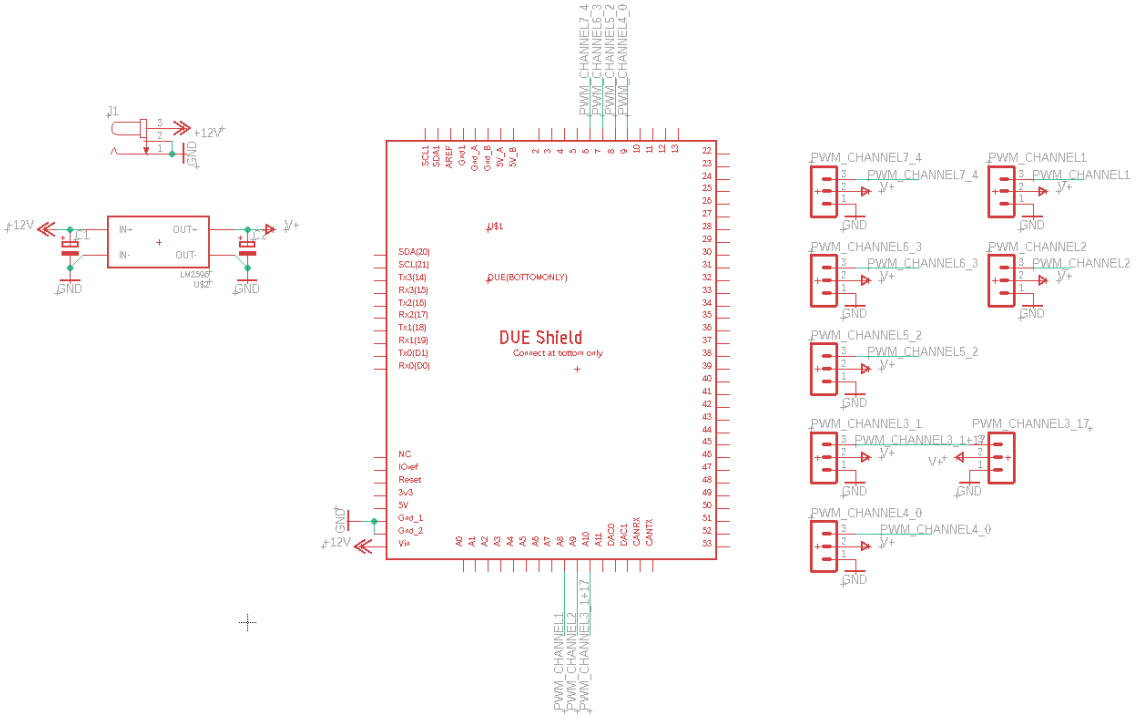


Obrázek 11: Pinout Arduina DUE

3.4 Linuxová distribúcia UDOObuntu 2

Pre UDOO QUAD sa vybral ako operačný systém oficiálna distribúcia linuxového systému UDOObuntu vo verzii 2. Počas práce na projekte bola vydaná novšia verzia 2.2, tá však neobsahovala kritické zmeny, preto nebolo treba updatovať. UDOObuntu je založené na operačnom systéme Ubuntu Trusty 14.04 LTS [9] a je špeciálne upravený firmou UDOO pre procesory s ARM architektúrou súčasne splňujúce hlavné ciele: rýchlosť a intuitívnosť.

Súčasne má tento špeciálne upravený systém veľkú „out-of-box“ podporu, čo znamená, že systém má v sebe predinštalované programy a knižnice pripravené pre vývoj na doske. Hlavnou prednosťou je napríklad už nainštalované plne funkčné Arduino IDE, ktoré je upravené pre komunikáciu s Arduinom DUE (základná verzia ho nepozná, musia sa dodatočne upraviť knižnice aj programátor) ako aj všetky potrebné Arduino knižnice. Ďalej bol systém upravený pre jednoduché ovládanie systémových Device Tree štruktúr dosiek UDOO pridaním takzvaného „Device Tree Editor“ či kontrolovanie aktuálneho stavu dosky pomocou nástroja „Web Control Panel“.



Obrázek 12: Navrhnutá schéma rozširujúcej dosky

Hlavné výhody UDOObuntu: [10]

- systém optimalizovaný pre dosiahnutie nízkych bootovacích časov a spotreby pamäte,
- LXDE grafický manažér pre lepší užívateľský pôžitok a menšiu záťaž hardvéru,
- ARMHF podpora, lepší výkon JAVA aplikácií,
- vysoký výkon sieťových zariadení a optimalizovaný sieťový výkon,
- najnovší kernel pre podporu externých zariadení ako kamery, herné ovládače, DVBT tunery, Kinect a veľa ďalšieho,
- Lubuntu software manažér a synaptic pre ľahkú správu balíkov,
- Python 3.4, OpenCV, Geany, Arduino IDE predinštalované ako vývojové nástroje,
- UDOO Configuration Tool pre ľahkú správu konfigurácie.

Súčasti pre správu UDOObuntu:

- ***Web Control Panel*** – je webová utilita špeciálne vytvorená pre jednoduchú konfiguráciu vývojovej dosky UDOO QUAD od bezdrôtového pripojenia po hostname a kontrolovanie statusu pripojenia a stavu celej dosky a všetkých vstavaných periférií. Súčasne umožňuje

užívateľovi naučiť sa, ako vytvoriť základné projekty a otestovať ich ako jednoduché Arduino sketche. Užívateľovi je dostupný vo forme webovej stránky po zadaní IP adresy dosky UDOO do prehliadača za predpokladu, že sú oba systémy pripojené k rovnakej sieti alebo priamo na doske vo výbere menu v sekcii *Preferences - UDOO Web Configuration.*

- ***Device Tree Editor*** – je webová aplikácia zameraná pre ovládanie hardvéru dosky UDOO. Je kompilovaná pomocou Device Tree compileru (dtc), ktorý produkuje binárne .dtb súbory. Dátová štruktúra je stromom jednotlivých uzlov a vlastností a môže obsahovať ľubovoľný druh údajov. UDOObuntu obsahuje niekoľko Device Tree blobov dostupných v umiestnení */dev/dts/.* Sú to rôzne kombinácie pre použité procesory a grafické výstupy použitej dosky. O načítanie správnej štruktúry sa stará boot loader pri štarte systému. Užívateľ má možnosť tieto štruktúry upravovať, nakoľko všetky piny sa spočiatku berú iba ako GPIO rozhrania, a tak si môže zvoliť ich nové funkcie. Môže takto spraviť znova vo forme webového rozhrania, ktoré znova nájde pod menu v sekcii *Preferences – Device Tree Editor.*

3.4.1 Prvé spustenie systému UDOObuntu

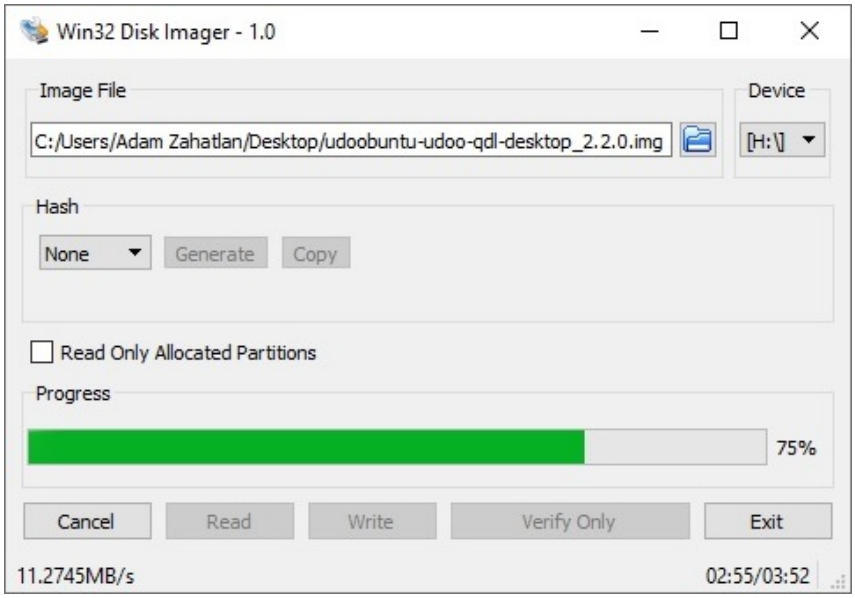
Prvou vecou, ktorú som musel vyriešiť, bolo ako dostať vývojovú dosku UDOO QUAD do funkčného štádia. UDOO QUAD má hneď niekoľko možností, akým spôsobom a aký systém spustiť a využívať k obsluhe dosky. Pre náš výber bola rozhodujúca kompaktnosť, rýchlosť, jednoduchosť a kompatibilita. Preto nedošlo k využitiu celkom rýchleho rozhrania SATA II prítomného na doske, ale k externej SD karte s bootovacím operačným systémom.

Požiadavkami pre správny nábeh a chod systému je správne sformátovanie SD karty, dostatočná veľkosť, oficiálny operačný systém a SD karta minimálne rýchlosti Class10 veľkosti väčšej ako 8 GB a zároveň nepresahujúcou 64 GB. Pokiaľ nie je použitá SD karta požadovanej triedy, nie je výrobcom zaručený správny chod a rýchlosť systému.

Ako prvé si užívateľ musí stiahnuť image systému UDOObuntu. Je tak možné spraviť na stránkach UDOO ako support pre dosku kúpenú užívateľom. Po stiahnutí a rozbalení ma užívateľ pripravený image. Ďalej musí užívateľ tento image dostať na SD kartu.

Nami bola zvolená SD karta *Samsung MicroSDXC* o veľkosti 64 GB. Spôsobov na nahratie imageu na kartu je hneď niekoľko. Tento image sa avšak na sformátovanú kartu nedá len nakopírovať ako v prípade inštalácie systému Windows. Napríklad mnou využívaný operačný systém Windows 10. Musel som si stiahnuť program *Win32DiskImager,* ktrorý je vidieť na obr. 13. Tento program slúži výhradne k nahrávaniu imageu na diskovú jednotku a je veľmi jednoduchý na obsluhu. Stačí si vybrať disk, ktorý sa sformátuje a nahrá sa naň image (v mojom prípade SD karta), zvolí sa nahrávaný image a spustí sa proces nahrávania tlačidlom *Write.* Po úspešnom nahratí môže užívateľ nahrané dáta ešte verifikovať a overiť tým ich správne nahratie.

Po úspešnom nahratí je SD karta rozdelená na celkom 3 partície a je pripravená k vloženiu do dosky UDOO. [9]



Obrázek 13: Program Win32DiskImager a nahrávanie obrazu na SD kartu

- 1 MB partícia rezervovaná ako úložisko pre boot-loader,
- 32 MB FAT partícia pripojená ako /boot, obsahujúca kernel, device tree štruktúry a dokumentáciu,
- EXT4 partícia slúžiaca ako root filesystem obsahujúca systém UDOObuntu.

Takto nahratý image je ihneď pripravený k používaniu a nie je potrebná jeho inštalácia. Systém sa nabootoval bez menších problémov a na obrazovke pripojenej cez HDMI vstup sa zobrazila plocha s logom UDOO. Počiatočný test ukázal, že systém je plne funkčný, fungovala WiFi, všetky potrebné periférie a aj Arduino IDE, ktoré už má v sebe niekoľko vzorov pre rôzne projekty.

3.5 Programové riešenie pre robotickú ruku

V nasledujúcich sekciách opíšem programové riešenie pre robotickú ruku. Toto riešenie je rozdelené na niekoľko sekcií, pretože bolo treba riešiť viac vecí, ktoré by sa nedali opísať v jednej sekcii. Každému problému sa tak venujem samostatne.

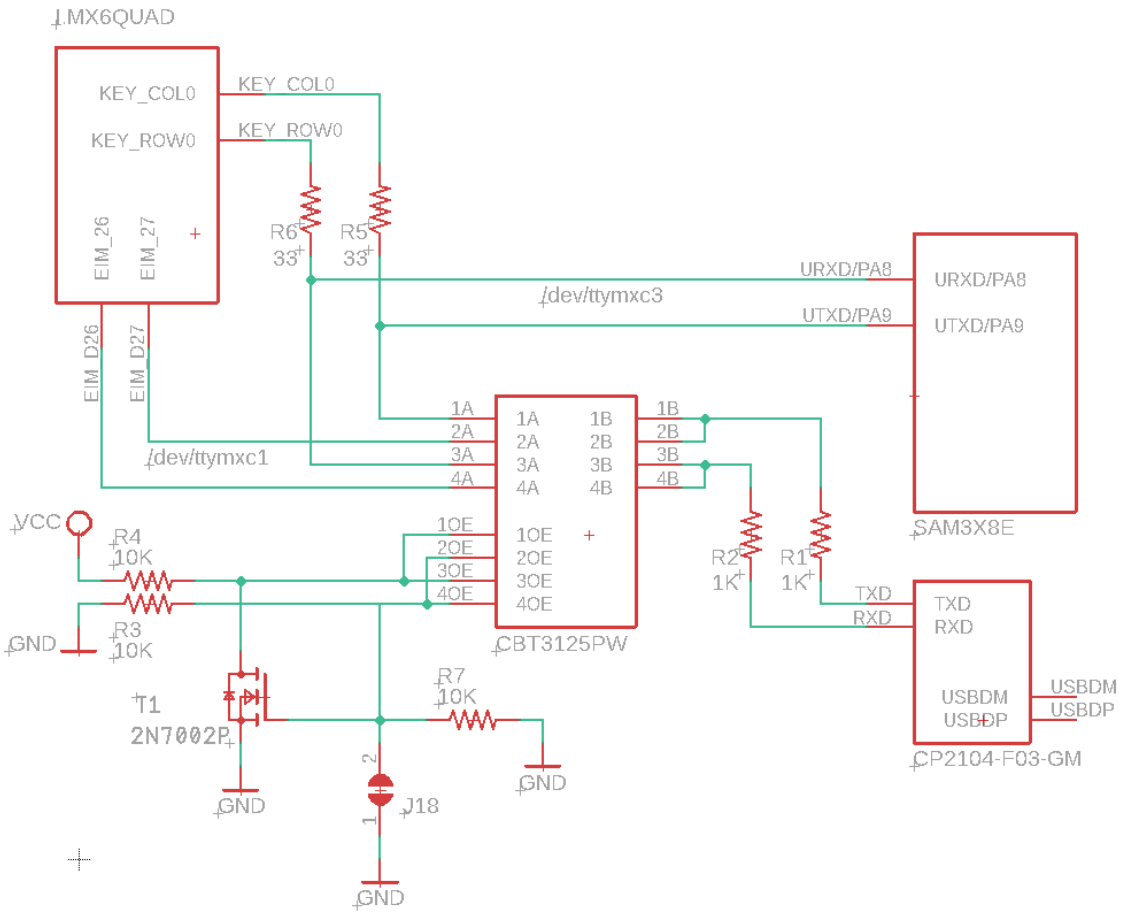
3.5.1 Medzi procesorová komunikácia

Tu nastala otázka, ako funguje medzi procesorová komunikácia medzi procesorom *i.MX6Quad* a *SAM3X8E* mikrokontrolérom. Preto som navštívil oficiálne stránky pre UDOO QUAD. Na stránkach a v dokumentácií bolo veľmi zle popísané a zobrazené prepojenie medzi oboma procesormi. Tieto popisy boli navrhnuté tak, aby aj nový užívateľ s minimálnymi znalosťami dokázal vedieť, ako asi táto komunikácia funguje. Pre naše potreby bol však tento opis nedostatočný.

Jediná vec, ktorá sa dala vyčítať bola, že existuje akási UART linka, pomocou ktorej procesory komunikujú na *baudrate 115200* a porte */dev/ttymxc3*. Súčasne existuje nejaký jumper, ktorým sa dala táto komunikácia prepínať a užívateľ mohol s týmito procesormi komunikovať aj z druhého počítača.

Preto bolo potrebné pozrieť sa na samotné zapojenie dosky UDOO QUAD [11] a určiť presne, čo sa v akej situácií presne v obvodoch dosky odohráva. Výsledkom preskúmania blokového schéma dosky UDOO QUAD je blokové schéma zobrazené na obr. 14.

Z tejto schémy sa už dalo presne určiť, čo sa deje v určitých prípadoch. Medzi procesorom *i.MX6Quad* a mikrokontrolérom *SAM3X8E* existuje UART linka zložená z dvoch kanálov *Rx* a *Tx* slúžiacich pre zápis a príjem dát. Táto linka je trvalá a tak sa nemôže stať, že by procesory medzi sebou ztratili možnosť komunikovať. Je označená ako */dev/ttymxc3*. Užívateľ však môže kontrolovať procesor, s ktorým chce komunikovať z externého zariadenia po linke označenej ako */dev/ttymxc1*. Na doske je k tejto linke pripojený konektor *J18*. Tento je v základnom stave prepojený jumperom a užívateľ po pripojení k Micro USB OTG môže komunikovať s procesorom



Obrázek 14: Blokové schéma komunikácie medzi procesorom i.MX6Quad a SAM3X8E mikrokontrolérom

i.MX6Quad. Môže vidieť celý boot systému, prihlásiť sa a pomocou konzole plne kontrolovať UDOObuntu. Po vykonaní krokov opísaných v kapitole 4.2 sa užívateľ dostane k mikrokontroléru *SAM3X8E,* s ktorým môže komunikovať, ale aj ho plne ovládať, premazať a nanovo nahrať do pamäte program. Toto je možné realizovať aj ručne, doska obsahuje niekoľko ďalších jumperov slúžiacich práve k tomuto účelu. Zmena komunikácie funguje vďaka tomu, že k UART linke prepojujúcej oba procesory je pripojený *CBT3125PW bus swich.* Ten mení smer roku dát Micro USB OTG portu nasledujúcim spôsobom:

- Bus switch prepína medzi portami *1A, 3A* a *2A, 4A,* ktoré sú prepojené medzi sebou a sú ovládané pomocou portov *1OE, 3OE* a *2OE, 4OE.*. Pokiaľ je na *OE* portoch privedený signál *HIGH (+5V),* sú tieto porty vypnuté a aktívne sú porty pripojené k *LOW (GND).* ako sa je možno dočítať v dokumentácií [12].
- V základnom stave je jumper *J18* zapojený. To znamená, že prepája *GND* s *Gate* konektorom *MOSFET tranzistora (2N7002).* Keďže je tento N-channel tranzistor v enhancement (obohacovacom) móde, je uzatvorený. Neprepája v tej chvíli nič a tak sú na *CBT3125PW bus switchi* aktívne porty *2A* a *4A,* pretože je na nich privedené *GND (0V).* Užívateľov PC je teda priamo spojený s *i.MX6Quad* procesorom po linke */dev/ttymxc1.*
- Pri odpojení jumpera *J18* dôjde k odpojeniu *GND* a tak sa tranzistor stane vodivým, čo má za následok otočenie polarity medzi *OE* vstupmi. To znamená odpojenie linky procesora *i.MX6Quad* a prepnutie pre linku mikrokontroléra *SAM3X8E.*

Ďalej je k tomuto switchu pripojený *CP2104-F03-GM I/O* ovládač rozhrania USB to UART bridge. Tento ovládač sa stará o prevod dát medzu UART a Micro USB OTG rozhraním, ktoré je k nemu pripojené.

Toto prepojenie sa nedá realizovať vždy. Linka medzi obomi procesormi je vytvorená pre dve zariadenia a tak pripojenie tretieho by mohlo vyústiť až k jej úplnej nefunkčnosti, pretože je založená na push-pull point-to-point protokole. Toto platí nielen pre pripojenie cez Micro USB OTG port, ale aj pre piny 0 a 1 Arduina DUE pripojených taktiež k tejto linke.

3.5.2 Arduino IDE/Atmel Studio IDP

Teraz, keď som presne zistil, ako funguje medzi procesorová komunikácia a odskúšal jej funkčnosť, bolo na čase začať vývoj riadiaceho programu pre robotickú ruku. Ten už síce bol vyvinutý autorom diplomovej práce, ktorá slúžila ako moja predloha, avšak tento program nie je prenositeľný na novozvolenú platformu UDOO QUAD a mikrokontrolér *SAM3X8E.* Preto som musel tento program znova navrhnúť a zaistiť jeho funkčnosť aj na tejto platforme.

Riadiaci program navrhnutý pre robotickú ruku má celkom jednoduchú funkčnosť. Ako som už naznačil v kapitole 2.2, autor minulého programu použil celkom dve riešenia k dosiahnutiu výsledku. Ja som sa rozhodol použiť v mojom riešení iba variantu riadenia servomotorov

pomocou PWM. Táto varianta je síce možno o niečo ťažšia na naprogramovanie kvôli správnej konfigurácií samotných PWM výstupov, no je viac rozšírená a zaručuje presné a spoľahlivé riadenie zapínania a vypínania portov na základe hodín, čím je teda presnejšie a správnejšie.

Pre návrh tohto riešenia som začal používať Arduino IDE zakomponované v UDOObuntu. Toto riešenie sa nakoniec javilo ako nedostatočné. Arduino IDE je skôr navrhnuté pre začiatočníkov, ktorým stačí, že mikrokontrolér naprogramujú. Arduino IDE totižto obsahuje plno funkcií, ktoré sa starajú o užívateľské pohodlie a uľahčujú jeho prácu.

Obhliadal som sa po inom IDE, ktoré má podporu mikrokontrolérov a súčasne dokáže skompilovať kód pomocou prekladača *armgcc,* ktorý používa Arduino IDE k skompilovaniu projektu pre *SAM3X8E* mikrokontrolér. Vedúci práce ma upozornil na Atmel Studio (súčasne vo verzii 7) [13] od spoločnosti Microchip Technology určené práve pre programovanie mikrokontrolérov na platforme Windows. Atmel Studio je IDP založené na Visual Studiu určené pre programovanie všetkých *AVR®* a *SAM* mikrokontrolérov v jazyku C/C++. Súčasne toto IDP obsahuje okrem podpory mikrokontrolérov príklady pre programovanie ich jednotlivých periférií, čo uľahčuje užívateľovi prácu.

3.5.3 FreeRTOS

Po preštudovaní dokumentácie k mikrokontroléru *SAM3X8E* a pozretiu príkladov pre jeho programovanie už len ostávalo vybrať RTOS, ktorý by zabezpečil správny chod programu. Je treba takto učiniť preto, že počas menenia pozície sa mikrokontrolér nachádza v akejsi kritickej sekcii, kedy je potreba dodržiavať správne rozširovanie šírky pulzu pre jednotlivé servomotory. Problém nastáva pri komunikácií cez UART linku, ktorá funguje asynchrónne. Mohla by nastať inkonzistenia pri vstupe dát, ich spracovávaní alebo rozširovaní už začatej zmeny šírky pulzu.

Mal som na výber použitie z dvoch RTOS systémov všeobecne overených a používaných pre mikrokontroléry: FreeRTOS [14] a Mbed. Po prečítaní niekoľkých porovnaní a zistení rozdielov medzi oboma operačnými systémami som sa rozhodol pre použitie FreeRTOSu. Rozhodol som sa hlavne kvôli tomu, že FreeRTOS sa javí sa ako spoľahlivý a ľahko použiteľný RTOS pre *i.MX6* platformu. Mbed podporuje menšiu základňu periférií a súčasne sa mi nepodarilo nájsť presnú zmienku, či tento RTOS funguje na *i.MX6* platforme spoľahlivo.

3.5.4 Nahratie programu na SAM3X8E mikrokontrolér

Po zvolení IDP a RTOSu bolo možné pustiť sa do vývoja samostatného programu. Narazil som ale na problém, že Atmel Studio síce plne podporuje mikrokontrolér *SAM3X8E* a aj vývojovú dosku Arduino DUE, neexistuje však oficiálne podporovaná cesta, ako tento program nahrať do pamäte procesoru. Ponúkli sa nám preto celkom 3 možnosti, ako tento program preniesť.

1. Modifikovať Arduino IDE aby sme videli presné kroky a príkazy, ktoré sa vykonávajú pri nahrávaní programu do pamäte mikrokontroléra a tieto príkazy externe volať zo sériovej linky pomocou druhého počítača.

2. Skompilovať program na externom počítači a takto skompilovaný program preniesť pomocou USB na dosku UDOO, odkiaľ by sa tento program nahral do pamäte mikrokontroléra.

3. Modifikovať Atmel Studio neoficiálnou cestou a môcť tak program nahrať do pamäte mikrokontroléra rovno z externého PC.

S vedúcim práce sme vyskúšali prvý krok a po modifikácií Arduino IDE sme videli presnú sekvenciu príkazov, ktoré sa volajú pri kompilovaní a nahrávaní programu.

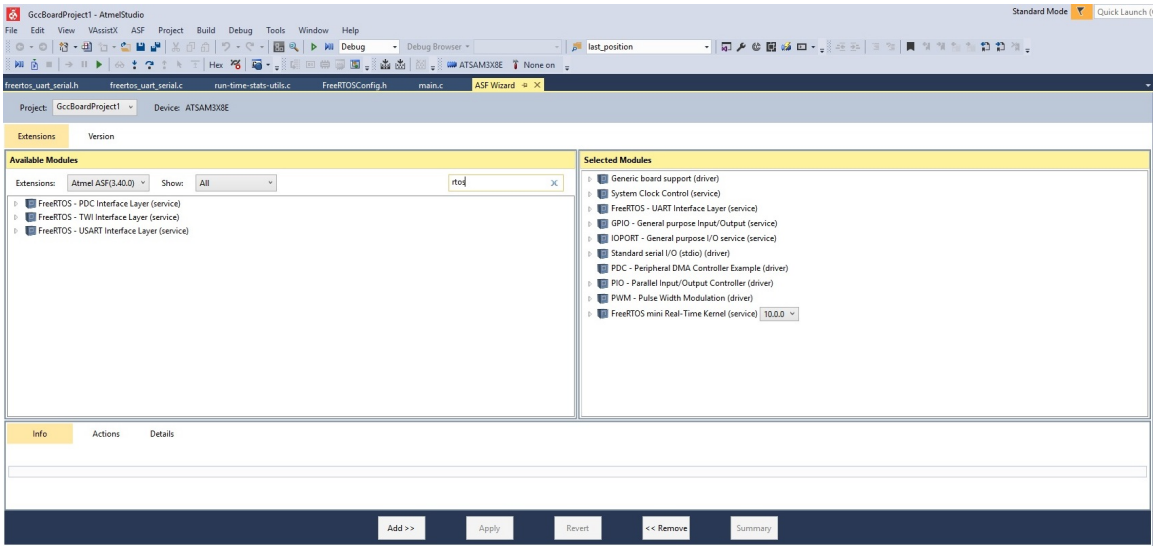
Vyskúšal som taktiež tretí krok, ktorý sa zdal najľahší a zároveň prinášal so sebou najmenej komplikácií. Atmel Studio má možnosť pridania vlastného programátora, užívateľ teda nemusí používať ich build-in programátor. Existuje niekoľko návodov ako tento programátor pridať, ako napríklad mnou vytvorený návod popísaný v prílohe A, čo sa mi nakoniec úspešne podarilo, test funkčnosti je možno vidieť v kapitole 4.3. Taktiež firma UDOO má na svojich stránkach postup, ako upraviť Arduino IDE externého počítača pre tieto účely. Práve úprava Arduina IDE pomocou *bossac.exe* programátora a *dll* knižnice je kľúčová pre pridanie programátora do Atmel Studia.

3.5.5 Návrh riadiaceho programu

Skompilovaním a nahratím jednoduchého example programu som si overil funkčnosť programátora ako aj korektnosť a funkčnosť nahratého programu, ako je vidieť v kapitole 4.3. Prešiel som teda na vývoj riadiaceho programu. Atmel Studio je intuitívne vďaka tomu, že je založené na Visual Studiu. Taktiež obsahuje *ASF Wizard,* ktorý užívateľovi umožňuje rýchlu a prehľadnú správu knižníc projektu ako je vidieť na obr. 15. Pre každý procesor sú k dispozícií knižnice pre všetky jeho periférie. Umožňuje taktiež importovanie FreeRTOS systému vo verzii vybranej užívateľom priamo do projektu, čo prináša výhodu a uľahčenie importovania RTOS pre môj riadiaci program.

Ako prvé som začal programovať PWM výstupy. V oficiálnych ukážkach programov nie je využívanie PWM rozhrania vysvetlené dobre a užívateľa môže zmiasť použitie len nejakých preddefinovaných `PIN_PWM_LEDX_CHANNELov`. Pozrel som sa preto bližšie do dokumentácie *SAM3X8E* mikrokontroléru. *PWML0 – PWML7* piny, ktoré nás zaujímali, boli v dokumentácií označené ako *PB16 – PB19* a *PC21 – PC24.* Takto označené porty som jednoducho našiel na pinout diagrame zobrazenom na obr. 11 a určil presne ich lokáciu na doske UDOO. Súčasne som sa po importovaní PWM driveru bližšie pozrel do hlavičkových súborov. Tu som už dokázal zistiť, čo sú `PIN_PWM_LEDX_CHANNELy` zač a ako sú nakonfigurované.

Pomocou tohto príkladu som sa snažil nakonfigurovať a vyskúšať základnú funkčnosť PWM výstupov, avšak márne. Použitím preddefinovaných funkcií na nastavenie PWM výstupov som nemohol dosiahnuť požadovaný výsledok. PWM porty fungovali nekonzistentne a nesprávne pri nakonfigurovaní viacerých portov naraz. V určitých kombináciách fungovalo PWM bezchybne. Nepodarilo sa mi zistiť, čo bolo príčinou zlé naprogramovanie preddefinovaných funkcií alebo



Obrázek 15: Atmel Studio 7, ASF Wizard

ich zlé použitie z mojej strany, preto som sa rozhodol o implementáciu vlastných funkcií, ktoré by mi zaisťovali inicializáciu PWM a ich následnú konfiguráciu. Výsledkom boli tieto funkcie:

- **`void init_pwms(uint8_t range_ms)`** – funkcia pre inicializáciu PWM portov. V tejto funkcii si vyberám hodiny, potrebné porty, nastavujem dobu opakovania generovania pulzu, takt hodín, spôsob generovania signálu a nakoniec zapínam samotné porty,
- **`void set_frequency(uint8_t channel, uint32_t duty)`** – funkcia pre nastavenie frekvencie PWM portu, kedy sa má generovať signál na výstupnom porte,
- **`void PWM_Handler(void)`** – funkcia slúžiaca ako handler eventov pre PWM. Mnou je táto funkcia nevyužitá, musí ale byť v programe pre správne fungovanie PWM definovaná,
- **`void PwmTask(void *pvParameters)`** – funkcia slúžiaca ako task pre FreeRTOS. V tejto funkcii sa vykonávajú hlavné výpočty šírky PWM pulzov v určitom čase pre všetky PWM kanály súčasne. Po výpočtoch sa tieto pulzy postupne počas daného času zväčšujú alebo zmenšujú, čo má za následok pohyb a kontrolovanie servomotorov robotickej ruky. Tento task má súčasne najvyššiu pridelenú prioritu ako najdôležitejší.

Takto naprogramované riešenie už fungovalo bez problémov a jednoduchým testom popísaným v kapitole 4.4 som si skontroloval správnosť generovania PWM signálu pomocou osciloskopu, ako je vidieť na obr. 20.

3.5.6 Sériová komunikácia

Ďalej bola na rade komunikácia cez UART linku. Tú som si už vyskúšal s Arduino IDE v kapitole 4.1, kde fungovala bezchybne. Pre kontrolovanie servomotorov sa využíva protokol bližšie popísaný v dokumentácií [2], ktorý vypadá následovne:

#0 P2100 #17 P1300 #1 P1300 #2 P433 #3 P1800 #4 P2433 T1000\r

Jedinou nevýhodou tejto komunikácie je, že tento vstup sa nedá prijať ako jeden celý string. UART linka prijíma postupne tieto znaky po jednom. Preto je treba tieto znaky postupne prijímať a ukladať do bufferu pre ich neskoršie spracovanie. Posledným znakom takto prijatého protokolu musí byť *„Carriage return“,* ktorý vypadá nasledovne: *'\r.'* Pokiaľ nie je na konci protokolu poslaný tento znak, sériová linka stále prijíma dáta a čaká na jeho poslanie.

Táto funkcia sa dala implementovať niekoľkými spôsobmi. FreeRTOS obsahuje framework *FreeRTOS+CLI,* ktorý uľahčuje správu komunikácie po UART linke. Užívateľ si môže zaregistrovať niekoľko príkazov naraz. FreeRTOS kontroluje vstup a vyberie správny zaregistrovaný príkaz. Táto funkcia nám nie je potrebná keďže sa bude po sériovej linke posielať defaultne len protokol. Takto som naprogramoval funkciu `create_uart_cli_task,` ktorá je použitá ako ďalší task FreeRTOSu, tentokrát s nižšou prioritou ako `PwmTask.` Beží v nekonečnej slučke, ktorá postupne prijíma znak po znaku, po prijatí *„Carriage returnu“,* vstup rozdelí, skontroluje správnosť a pridá do `Queue` pre spracovanie `PwmTaskom.`

Po úspešnom naprogramovaní som si overil funkčnosť komunikácie, správnosť spracovania protokolu a nastavovania šírky PWM pulzov počas daného času jednoduchým testom. 4.5 Súčasne som vyskúšal funkčnosť rozširujúcej dosky, ktorú som opísal v kapitole 3.3.

3.6 Aplikačné riešenie pre robotickú ruku

Rovnako ako riadiaci program, aj aplikačné riešenie pre robotickú ruku je rozdelené do niekoľkých sekcií. Taktiež bolo treba riešiť viacero problémov, ako aj súčasne opísať funkcionalitu už dodanej aplikácie RoboticArm, aby bolo mnou upravené riešenie bližšie pochopiteľné.

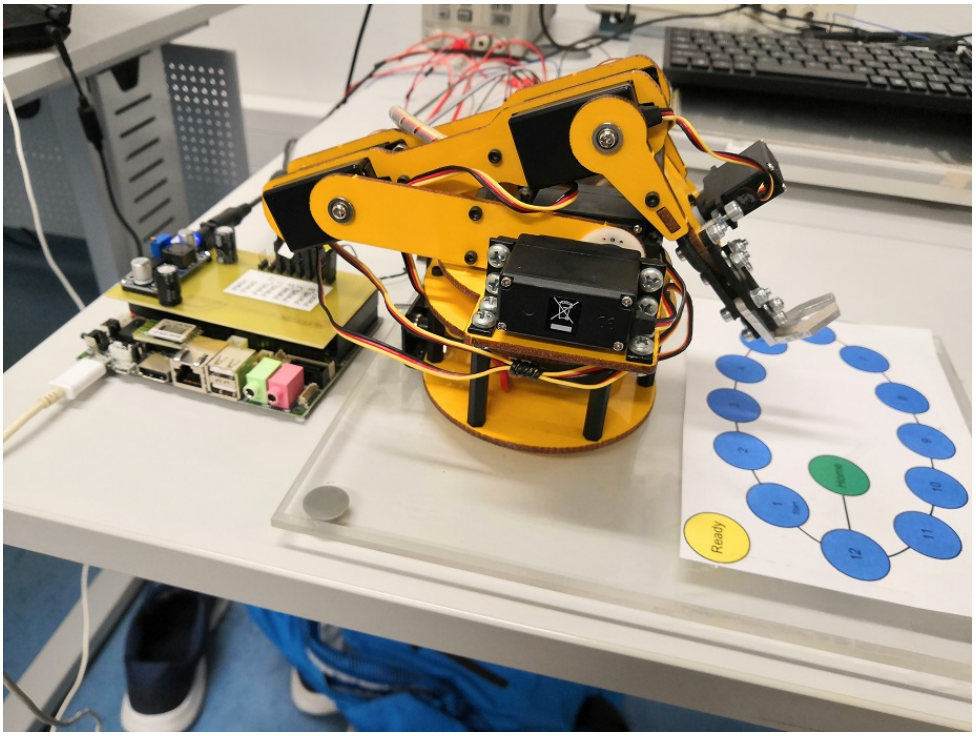
3.6.1 Prenos aplikácie RoboticArm

Ďalším krokom po úspešnom vytvorení riadiaceho programu pre robotickú ruku bolo overenie jeho funkčnosti. Tú som síce overil testom popísaným v kapitole 4.5, ešte však bolo treba zistiť, či program funguje korektne pod desktopovou aplikáciou RoboticArm. Jednotlivé kroky potrebné k spusteniu aplikácie sú popísané v prílohe C.

RoboticArm je už upravená autorom minulého projektu. Ako prvé som musel upraviť túto inicializačnú premennú `mInterface` definovanú v triede `roboticarm.cpp` na `/dev/ttymxc3,` čo je UART linka spojujúca oba procesory *i.MX6Quad* a *SAM3X8E.* Aplikácia naviaže spojenie po sériovej linke `init` funkciou, v ktorej sa preposielajú dva parametre: *názov linky* a *baudrate.* Pokiaľ bolo toto spojenie úspešne naviazané, aplikácia pošle *„request“ na virtuálny kanál 7* po sériovej linke, čím si chce zistiť poslednú polohu robotickej ruky. Pokiaľ je jej odpoveď *„noInit“,* nastaví sa do určitej základnej polohy. Túto polohu som si predefinoval na „parkovaciu“ pozíciu ukázanú na obr. 16, ktorú som si sám určil. Pokiaľ bol odpoveďou protokol, aplikácia prijala

správu, rozparsovala si jednotlivé časti a uložila posledné nastavené polohy servomotorov robotickej ruky. Toto nastavenie zaistilo ochranu proti náhodnému pádu aplikácie, kedy nebola známa posledná poloha robotickej ruky. Ďalej bolo treba upraviť *konštruktor* v triede `camera.cpp`. Tento konštruktor pri inicializácií nastavuje hodnotu `cv::VideoCapture(1)` na fixnú hodnotu zariadenia *video1.* UDOO QUAD však po pripojení externej USB kamery Genius vytvorí ďalšie zariadenie *video2,* ktoré je pripojená kamera. Stane sa tak preto, že zariadenia *video0* a *video1* sú už rezervované systémom pre integrované rozhranie kamery. Preto som nastavil hodnotu v konštruktore na *2.*

Takto upravená aplikácia bola plne pripravená pre skúšku mnou vytvoreného riadiaceho programu. Hlavným bodom bolo vyskúšanie funkčnosti triedy `Game2,` ktorá predstavovala zjednodušenú variantu hry *„Človeče, nehnevaj sa!“.* Kamera úspešne rozoznávala pomocou algoritmov pre spracovanie obrazu implementovaných v knižnici *OpenCV* hodnotu hodenú na hracej kocke, ktorá sa premietla do programu. Figúrka stojaca na políčku sa musela o túto hodnotu posunúť, preto program vybral korektné súradnice bodu načítané zo súboru `GamePositions,` vypočítal šírky pulzov pre jednotlivé servomotory, zložil ich do definovaného protokolu a poslal po sériovej linke */dev/ttymxc3* mikrokontroléru *SAM3X8E.* Ten si správu zanalyzoval a začal so zmenou šírky pulzov na doručené hodnoty. Nakoľko hra fungovala skoro bezchybne okrem občas nesprávneho umiestnenia figúrky mimo jej miesto o cca 0,5 cm, o ktorom písal aj autor predchádzajúceho riešenia v svojej práci, môžem považovať funkciu riadiaceho programu riadeného aplikáciou RoboticArm za správnu.



Obrázek 16: Robotická ruka fungujúca na vývojovej doske UDOO QUAD

3.6.2 Úpravy aplikácie RoboticArm

Po overení funkčnosti riešenia som mohol začať s implementáciou vlastnej hry. Po konzultácií s vedúcim práce ohľadom detailov malo ísť znova o zjednodušenú variantu hry *„Človeče, nehnevaj sa!“*. V tejto verzii hry si mal užívateľ nakresliť vlastnú dráhu podľa svojej predstavy.

S ohľadom na veľkosť displeja, maximálnu dĺžku ramena som zvolil základnú veľkosť kresliacej plochy pre dráhu. Súčasne som upravil veľkosť aplikácie na maximálne rozlíšenie dotykovej plochy, aby sa využil maximálny potenciál kreslenej dráhy. Jednotlivé okná aplikácie sú riešené ako záložky. Prepínanie medzi týmito záložkami zmení súčasné okno. Rozšíril som preto aplikáciu o ďalšiu záložku `„Draw Game".` Z pôvodnej hry som prevzal umiestnenie grafického znázornenia kocky, tlačidlá a výstup z kamery. Umiestnil som hraciu plochu tak, aby ruka nemala problém s maximálnou dĺžkou ramena. Ďalej som vytvoril ďalšie tlačidlo pre vyvolanie kreslenia novej dráhy.

Samotné kreslenie hracej plochy som riešil cez triedu *QPainter*. Táto trieda je súčasťou *Qt* knižnice a je navrhnutá pre jednoduché a rýchle kreslenie geometrických útvarov. Pomocou nej vykresľujem kruhy predstavujúce pozície pre figúrku, tak aj kreslenú dráhu. Užívateľ túto dráhu bude kresliť ťahaním prstu po dotykovej ploche. Na to, aby som ju zachytil, použil som eventy `mousePressEvent, mouseMoveEvent` a `mouseReleaseEvent` triedy `QMouseEvent.` Mouse eventy sa hodia perfektne, nakoľko dotyky po dotykovej ploche simulujú stlačenie hlavného tlačidla myši. Počas toho, ako užívateľ kreslí dráhu, kontroluje sa vzdialenosť pridaných bodov. Takto nemôže dôjsť ku kolízií jednotlivých vygenerovaných políčok. Ďalej som pridal kontrolu, aby užívateľ nezašiel s kreslením dráhy mimo hraciu plochu, čím by nakreslil bod mimo vyhradenú plochu. Taktiež je treba kontrolovať, aby užívateľ nezačal alebo neskončil s kreslením dráhy moc blízko k počiatočnému bodu natvrdo zvolenému v ľavom hornom rohu kresliacej plochy. Po neúspešnom nakreslení dráhy je plocha resetovaná a užívateľ môže kresliť od začiatku. Pokiaľ bola plocha dobre nakreslená, uloží sa do súboru `drawGame.png,` uzavrie sa widget s kreslením plochy a pozície vygenerovaných bodov sú uložené do súboru `DrawGamePositions.` Takto uložené súradnice a hracia plocha zaistia konzistenciu pri náhodnom páde aplikácie RoboticArm.

Túto zmenu dráhy bolo treba z aplikácií premietnuť. Pre jej korektné zobrazovanie som musel najprv upraviť triedu `mygraphicsview.cpp.` Vytvoril som metódu `loadPoints(QString defaultPath),` ktorá načítava súradnice vygenerovaných bodov zo súboru `DrawGamePositions.` Ďalej som vytvoril metódu `reloadTrack()` v triede `roboticarm.cpp,` ktorá sa stará o znovu načítanie obrázku `drawGame.png` s upravenou hracou plochou a taktiež načítanie pozícií metódou `loadPoints(QString defaultPath).`

Po testoch opísaných v kapitole 4.6 som si overil, že dráha sa generuje a ukladá korektne, rovnako ako aj pozičné body. Ďalej bolo treba počítať pozície bodov na ploche, aby robotická ruka vedela, kam má figúrku umiestňovať. Na to, aby sme boli schopní toto počítať, musíme

najprv zistiť, aké matematické výpočty sa vykonávajú s načítanými súradnicami v pôvodnej hre `Game2.`

3.6.3 Matematické výpočty aplikácie RoboticArm

Každá zmena polohy pre každý servomotor musí byť najprv spracovaná a prepočítaná hneď v niekoľkých krokoch:

1. Vstupom pre výpočty sú hodnoty *X, Y, Z* a *A,* pričom *X* a *Z* sú súradnice v priestore robotickej ruky (cylindrická súradnicová sústava), *Y* je uhol natočenia základne robotickej ruky a *A* je uhol naklonenia úchopnej časti ramena robotickej ruky. Tento posledný uhol by mal byť optimálne vždy 90° k ploche dráhy. Tieto vstupné dáta je však nutné konvertovať kvôli tomu, že jednotlivé servomotory sú kontrolované krokmi.

   $$H = 6, L1 = L2 = 10, L3 = 6\,[cm]$$

   Toto sú dĺžky jednotlivých častí robotickej ruky znázornené zeleno na obr. 3. *H* označuje výšku základne. *L1 - L3* sú dĺžky ramien robotickej ruky. *L1* je rameno spojené so základňou, *L2* je rameno spojené s ramenom *L1* a *L3* je úchopné rameno spojené s ramenom *L2.* Výpočty vyzerajú nasledovne:

   $$X_b = \frac{X - L3 \times \cos\left(A * \pi \,/\, 180\right)}{2 \times L12} \tag{1}$$

   $$Z_b = \frac{Z - H - L3 \times \cos\left(A * \pi \,/\, 180\right)}{2 \times L12} \tag{2}$$

   $$Q = \sqrt{\left(\frac{1}{{X_b}^2 + {Z_b}^2) - 1}\right)} \tag{3}$$

   $$P1 = \arctan\left(\frac{Z_b + Q \times X_b}{X_b - Q \times Z_b}\right) * 180 \,/\, \pi \tag{4}$$

   $$P2 = \arctan\left(\frac{Z_b - Q \times X_b}{X_b + Q \times Z_b}\right) * 180 \,/\, \pi \tag{5}$$

   $$T1 = P1 - 90 \tag{6}$$

   $$T2 = -(P2 - P1) \tag{7}$$

   $$T3 = A - P2 \tag{8}$$

T1 - T3 sú vypočítané uhly, ktoré je však ešte treba znormalizovať:

$$T1 = T1 - \frac{T1}{360} \times 360 \tag{9}$$

$$T2 = T2 - \frac{T2}{360} \times 360 \tag{10}$$

$$T3 = T3 - \frac{T3}{360} \times 360 \tag{11}$$

T1 - T3 sú teraz znormalizované uhly, ktoré určujú sklon proti základni, prvému a druhému ramenu robotickej ruky. Uhol otočenia základne počítať netreba, nakoľko už je predaný ako hotový uhol.

2. Tieto vypočítané hodnoty je treba previesť na jednotlivé kroky servomotorov robotickej ruky výpočtami uvedenými vo výpise 1.

   Tieto výpočty sa môžu zdať mätúce, po bližšom preskúmaní triedy `senddata.cpp` sa vysvetlí spôsob počítania. Keďže je každé servo na ruke prichytené v odlišnom uhle a mieste, je treba korekcie, ktorá určí stredovú pozíciu servomotora, ako aj jej prepočet na jednotlivé kroky voči uhlu. Každé servo má taktiež špecifický prepočet *STEP_TO_ANGLE,* ktorý určuje šírku pulzu servomotora k vypočítanému uhlu. Všetky tieto korekcie je možno vidieť v tabuľke 2. Takto prepočítané kroky už sú použiteľné pre riadenie servomotorov. Ďalej je potreba tieto kroky poslať po sériovej linke mikrokontroléru *SAM3X8E* k spracovaniu.

3. Nakoľko sa z pôvodného projektu, ktorý dodala firma NXP uchovali čísla kanálov, uchoval sa aj protokol, s ktorým sa s týmito servomotormi komunikovalo [2]. Inak by to viedlo k značnému zásahu do aplikácie RoboticArm. Správny formát komunikačného protokolu vypadá nasledovne:

   #0 P2100 #17 P1300 #1 P1300 #2 P433 #3 P1800 #4 P2433 T1000\r

   Z uvedeného príkladu je možno vidieť, že na to, aby bola správa prijatá a spracovaná aplikáciou RoboticArm, musí byť v nasledovnom formáte:

   #id_X Pšírka_pulsu_X #id_Y Pšírka_pulsu_Y ... Tdoba_presunu\r

   - ***#id*** sú kanály servomotorov uvedené v tabuľke 1, teda celkovo 6. Pokiaľ sa zadáva kanál 1 alebo kanál 17, musí byť zadaný aj jeho párový kanál, nakoľko sú servá spriahnuté a rozličné nastavenie by vyústilo k nepredvídateľnému chovaniu. Tento parameter musí byť zadaný vždy pred parametrom ***Pšírka_pulsu_X***.
   - ***Pšírka_pulsu_X*** určuje šírku pulzu odosielanú na daný kanál servomotora. Šírka pulzu sa odosiela v mikrosekundách. Ako bolo spomenuté v kapitole 2.3, šírky pulzu sa budú odosielať v rozmedzí 500 - 2500 us.

- Posledným parametrom je ***Tdoba_presunu.*** Tento parameter určuje dobu, za ktorú sa majú všetky servomotory dostaviť na svoju konečnú polohu, to znamená zadanú šírku pulzu pre každý kanál. Tento parameter nemusí mať pevné miesto, a tak môže byť umiestnený na hociakom mieste v protokole.

4. Takto zložený príkaz je pripravený na odoslanie riadiacemu mikrokontroléru *SAM3X8E* po linke */dev/ttymxc3,* ktorý obsluhuje PWM porty pre robotickú ruku. Má taktiež za úlohu si skontrolovať správnosť prijatých dát.

O všetky tieto výpočty sa starajú triedy *game2.cpp* a *senddata.cpp* aplikácie RoboticArm.

```
VAL = VAL + CENTER_ANGLE
VAL = CENTER_STEPS + ( VAL * STEP_TO_ANGLE)
```

Výpis 1: Pseudokód pre prevod uhlov na šírku pulzov servomotorov

Servo	CENTER_ANGLE	CENTER_STEPS	STEP_TO_ANGLE
1	0°	500	10
2	90°	633	10
3	42°	503	10
4	90°	560	10
5	0°	1119	11.6

Tabulka 2: Vyčítané parametre triedy senddata.cpp pre prevod uhlu na kroky servomotora

3.6.4 Vlastné výpočty pre kreslenú dráhu

Pri znalosti, ako prebiehajú výpočty v aplikácií RoboticArm som mohol začať s výpočtami pre našu dynamicky kreslenú dráhu. Mohol som tak spraviť po umiestnení robotickej ruky na sklo k dotykovej obrazovke. Vedel som že je potreba vzdialeností *X, Z* a uhlov *Y, A.* Súčasne som poznal len veľkosť hracej plochy, veľkosť LCD dotykového panelu, a súradnice jednotlivých kreslených bodov. Navrhol som teda výpočty, ktoré mi tieto parametre prevedú na potrebné hodnoty.

1. Jednotlivé výpočty sa musia pre presnosť realizovať v milimetroch. Preto vypočítané *DPI (dots per inch)* LCD displeja bolo nutné previesť z pixelov na milimetre nasledovne:

$$DPI = \frac{1366\,px}{(344\,mm/25,4\,mm\,per\,inch)} \tag{12}$$

$$DPM = \frac{DPI}{25,4\,mm} \tag{13}$$

1366 px je rozlíšenie LCD displeja na jeho šírke *344 mm. 25,4* je počet milimetrov na jeden palec (inch). Týmto výpočtom som dostal jetnotku *DPM (dots per milimeter)* vytvorenú pre túto prácu, s ktorou som si mohol každú súradnicu v pixeloch prepočítať na milimetre.

2. Teraz som si musel zmerať posun robotickej ruky voči displeju. Tieto vzdialenosti je potreba počítať od stredu základne. Tá má základňu o priemere *9.6 cm,* čiže ku každému meraniu bolo treba pridať polomer *48 mm.* Na X-ovej osi je tento rozmer daný posunutím od kraju displeja. Na Y osi však základňa kvôli krátkej dĺžke ramena zasahuje do plochy displeja, a tak bude treba túto odchýlku zohľadniť pri nasledujúcich výpočtoch.

3. Ďalej bolo treba zadefinovať konštanty. Už som vedel pozície robotickej ruky k displeju, teraz bolo treba vypočítať jednotlivé rozmery z aplikácie pre hraciu plochu. Pomocou funkcie `mapToGlobal(QPoint(0, 0))` som si zistil globálne súradnice widgetu. Výsledkom boli konštanty zobrazené vo výpise 2.

4. Všetko bolo tak pripravené pre počítanie súradníc a uhlov. Po načítaní súradníc bodov hracej plochy zo súboru `DrawGamePositions` mnou vytvorenou funkciou `loadPoints(QString DefautlPath)` som vytvoril funkciu `calculateCalibrationPoints(),` ktorá vykonáva nasledujúce výpočty:

$$dx = ARM\_POS\_X - \left(\frac{X\_CORRECTION + positionPoints[i].x()}{DPM}\right) \tag{14}$$

$$dy = ARM\_POS\_Y - \left(\frac{Y\_CORRECTION + positionPoints[i].y()}{DPM}\right) \tag{15}$$

$$d = \sqrt{(dx^2 + dy^2)} \tag{16}$$

$$angleA = -\left(\tan\left(\frac{dx}{dy}\right) * 180 \,/\, \pi\right) \tag{17}$$

$$c = round(d) - DRAWGAME\_FPOS\_X - DRAWGAME\_FPOS\_Z \tag{18}$$

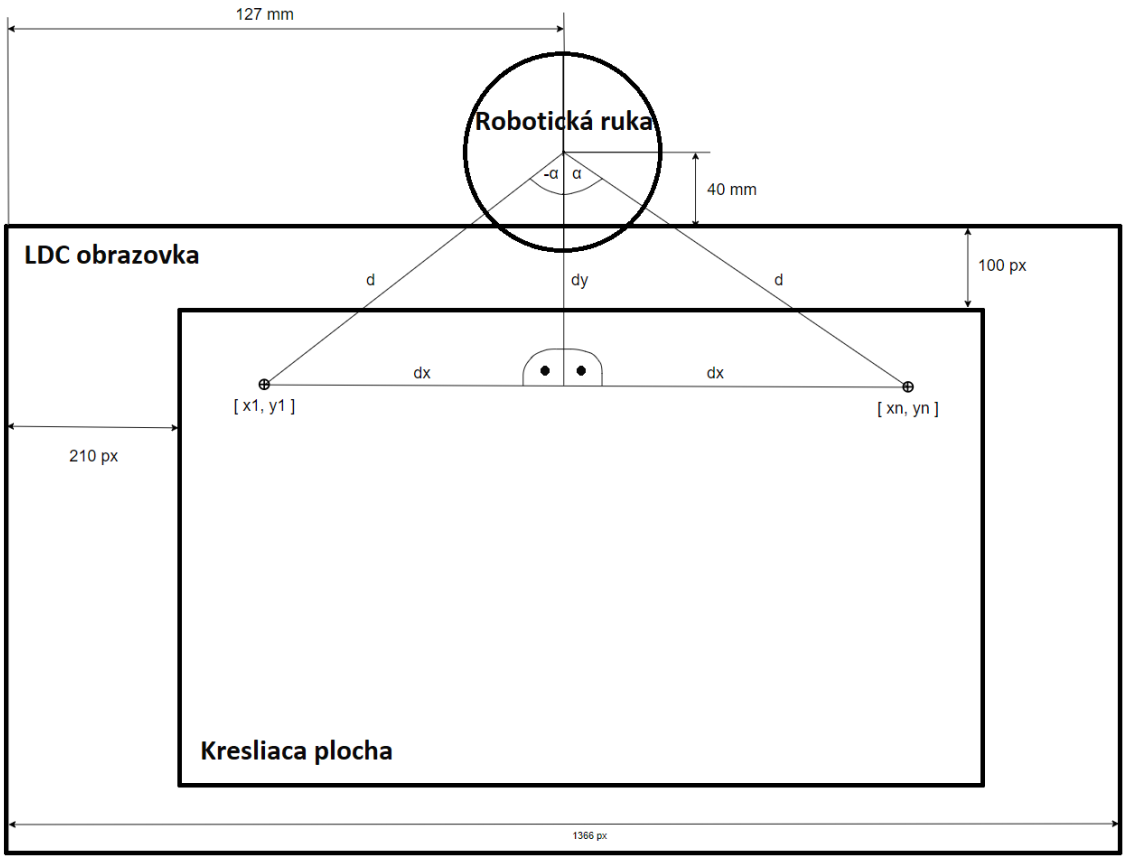
$$alfa = round(angleA) \tag{19}$$

$$z = -\left(\frac{round(d) - ARM\_POS\_X + BASE\_ADVANCE - 10}{10}\right) \tag{20}$$

c, alfa a *z* boli výsledné hodnoty, ktoré sa zapísali do poľa `CalibratingPositions[]` a následne sa predali funkcii `SolveCoordinates(x, y, z, angle)` ako parametre pre posun ruky.

Správnosť navrhnutých výpočtov navrhnutých na obr. 17 a presnosť umiestňovania som si overil pomocou jednoduchého testu opísanom v kapitole 4.7 na mnou nakreslenej testovacej dráhe.

Žiaľ tento test taktiež odhalil veľké obmedzenie zo strany robotickej ruky. Pri skúšaní hry dvoch hráčov dochádzalo k zrážaniu figúrok, keď boli umiestnené pozičné body sa sebou v rovine s robotickou ruku. Na vine bol fakt, že tá je veľmi nemotorná, nepresná a jej úchyt nie je navrhnutý s účelom manipulovať s uchyteným objektom v blízkosti iných objektov. Vyriešiť by sa to dalo zvýšením vzdialenosti medzi vykreslenými bodmi, čo by však prinieslo obmedzenie vo veľkosti a čase, za ktorú by sa táto dráha dala prejsť, nakoľko by bolo týchto bodov veľmi málo. Preto, aby nedošlo k zníženiu kvality práce, sa od hry dvoch hráčov nakoniec po dohode s vedúcim práce upustilo.

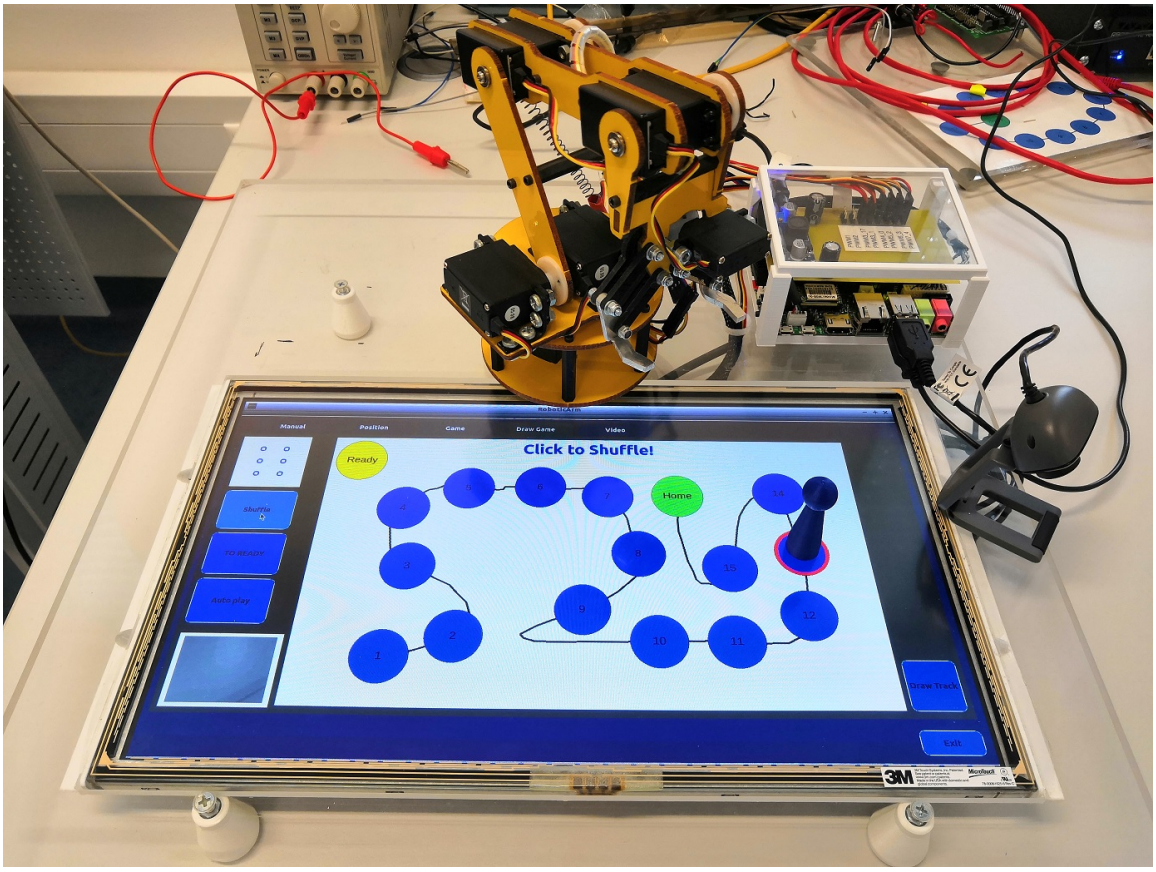


Obrázek 17: Nákres s výpočtami pre kreslenú dráhu

Hotový systém je nakoniec umiestnený na sklenenom panelu. Na ňom je prítomná aj doska UDOO QUAD, LCD dotykový displej, robotická ruka a kamera pre snímanie kocky. Držiaky pre LCD displej, modul dotyku, dosku UDOO a kameru boli vymodelované a vytlačené na školskej 3D tlačiarni. Celý tento systém je možno vidieť na obr. 18.

```
DRAWGAME_FPOS_X 50 //základná dĺžka ramena v mm
DRAWGAME_FPOS_Y 70 //základný uhol natočenia základne v stupňoch
DRAWGAME_FPOS_Z 53 //základná výška úchopného ramena v mm
ARM_POS_X 172 //stredová pozícia ruky od kraja obrazovky na ose X v mm
ARM_POS_Y -40 //stredová pozícia ruky od kraja obrazovky na ose Y v mm
X_CORRECTION 210 //odsadenie obrázku dráhy na ose X v pixeloch
Y_CORRECTION 100 //odsadenie obrázku dráhy na ose Y v pixeloch
DPI (1366/(344/25,4)) //prevod rozmerov obrazovky na DPI
DPM (DPI/25,4) //vypočítané DPM pre prevod pixelov na milimetre
BASE_ADVANCE 32 //odsadenie stredu základne, s ktorým počítajú prepočty
    programu
```

Výpis 2: Konštanty triedy drawgame.cpp pre kreslenú dráhu



Obrázek 18: Hotový systém ukazujúci svoju funkčnosť

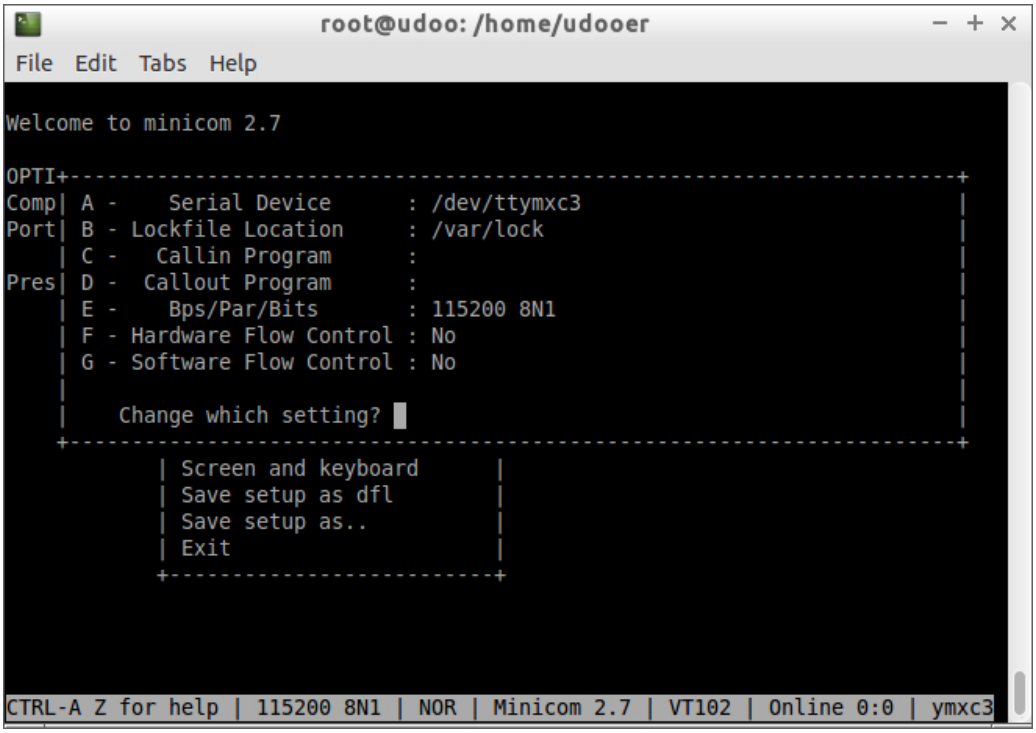
4 Test modifikovaného riešenia

4.1 Test medzi procesorovej komunikácie

Ako prvý test som vykonal test medzi procesorovej komunikácie. Táto funguje medzi procesorom *i.MX6Quad* a *SAM3X8E* mikrokontrolérom. Komunikácia je nadviazaná na UART linke označenej ako */dev/ttymxc3*. Medzi procesormi sa nadviaže po spustení *ubootu*. Vytvoril som si jednoduchý sketch v Arduino IDE pre skontrolovanie funkčnosti komunikácie. Ako prvé je treba inicializovať baud rate portu príkazom *stty:*

udooer@udoo: $sudo stty -F /dev/ttymxc3 cs8 -crtscts 115200

Ďalej môže používateľ komunikovať po sériovej linke pomocou dvoch terminálov a príkazov *echo/cat* alebo použiť program minicom určený pre tieto účely. Najprv je ale treba vypnúť hardware flow control ako na obr. 19 v nastaveniach serial portu minicomu. Možno tak vykonať po jeho spustení pomoocou klávesovej skratky *CTRL + A a O.*



Obrázek 19: Modifikácia sériovej linky programu minicom

Teraz stačilo spustiť program minicom nasledujúcim príkazom

udooer@udoo: $sudo minicom -D /dev/ttymxc3 -b 115200

napísať niečo na sériovú linku a potvrdiť ako vidieť na výpise 3.

```
dsa1864
Hi, this just came: dsa1864
Ahoj, ako sa mas?
Hi, this just came: Ahoj, ako sa mas?
+ľščťžýá
Hi, this just came: +ľščťžýá
```

Výpis 3: Výpis zo sériovej linky programu minicom pre /dev/ttymxc3

4.2 Test medzi procesorovej kominukácie z externého počítača

Ako ďalšie som vyskúšal komunikáciu po Micro USB OTG sériovej linke označenej ako */dev/ttymxc1,* aby som vyskúšal jej funkčnosť pre programovanie z externého PC. V defaultnom stave je jumper *J18* zapojený, užívateľ vidí *uboot* a komunikuje po sériovej linke s procesorom *i.MX6Quad* ako na výpise 4.

```
U-Boot 2015.10-00061-g68849f9 (Jan 08 2016 - 17:01:43 +0100)

CPU:   Freescale i.MX6Q rev1.5 at 792 MHz
Reset cause: POR
Board: UDOO Quad
DRAM:  1 GiB
MMC:   FSL_SDHC: 0

...

Ubuntu 14.04.6 LTS udoo ttymxc1

udoo login: udooer
Password:
Last login: Wed Apr 17 21:53:49 UTC 2019 on ttymxc1
Welcome to Ubuntu 14.04.6 LTS (GNU/Linux 3.14.56-udooqdl-02044-gddaad11 armv7l)

 * Documentation: http://www.udoo.org/docs/
udooer@udoo:~$ uname -a
Linux udoo 3.14.56-udooqdl-02044-gddaad11 #3 SMP PREEMPT Tue Dec 6 16:26:55 UTC
    2016 armv7l armv7l armv7l GNU/Linux
```

Výpis 4: Výpis zo sériovej linky pri pripojenej k procesoru i.MX6Quad

Pokiaľ chce užívateľ komunikovať s mikrokontrolérom *SAM3X8E,* je to trocha zložitejšie a treba vykonať nasledujúce kroky:

1. Pripojiť za k sériovej linke /dev/ttymxc1 pomocou Micro USB OTG a pripojiť UDOO QUAD k napájaniu.
2. Prerušiť spustenie ubootu počas 3 sekundovej pauzy bootovania.
3. Odpojiť jumper J18.

Teraz má užívateľ plný prístup a môže komunikovať s mikrokontrolérom *SAM3X8E.* K testu som využil predchádzajúci sketch už nahratý do pamäte procesora.

4.3 Test externého programátora pre Atmel Studio

Po úspešnom teste medzi procesorovej komunikácie a nasadení nového programátora som mohol vyskúšať jeho funkčnosť. Ako som už spomenul v kapitole 3.5.2, Atmel Studio obsahuje pre všetky podporované procesory vzorové príklady ukazujúce ich funkčnosť.

Pre test som si vybral ukážku *PWM_LED_EXAMPLE* pre mikrokontrolér *SAM3X8E.* Po jeho úspešnom skompilovaní som sa pokúsil program nahrať do pamäte procesora. Po uskutočnení krokov opísaných v teste 4.2 som zistil, že program sa nahrať nedá. Vždy nahrávanie skončilo na hláške

No device found on COM8
chipId=0x30303030

Začal som teda hľadať, či je chyba v komunikácií alebo mnou vytvoreným externým programátorom. Príčina sa avšak vyjasnila po prečítaní oficiálnej UDOO dokumentácie. Tu sa zmieňujú, že pre úspešné nahratie programu z externého počítača je treba bežiaci *kernel.* Ten som ja zastavil zrušením spustenia *ubootu.* Je tomu tak preto, že externý programátor si volá jednotlivé príkazy uložené práve v kernelu a tak dôjde k úspešnému nahratiu programu. Nechal som načítať *uboot* a po spustení systému som odpojil jumper *J18* a skúsil program nahrať znova.

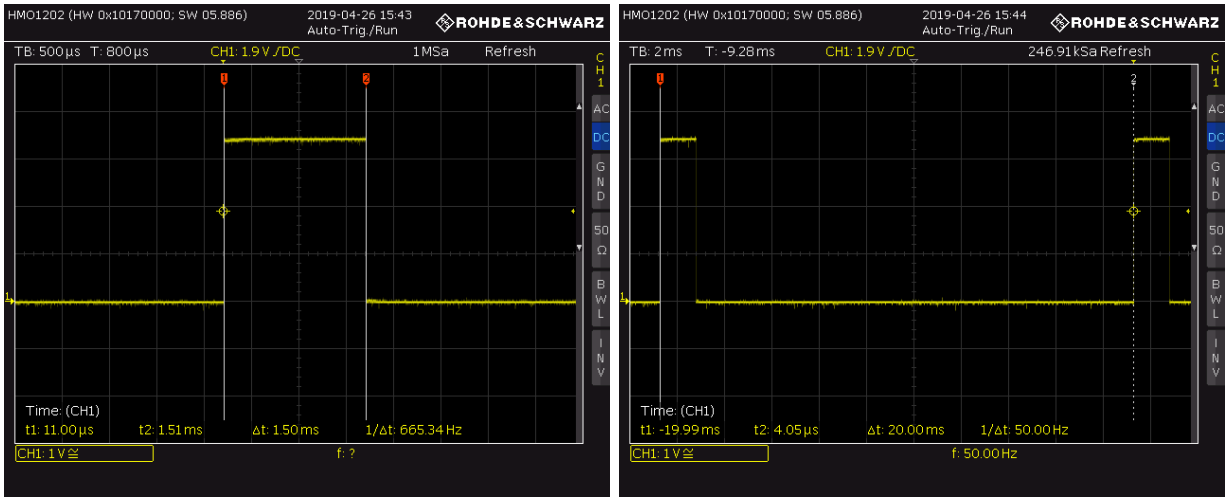
Vo výpise 5 je možno vidieť volanie *bossac.exe* súboru upraveného Arduina IDE ako aj priebeh nahrávania a kontrola správnosti nahratia programu. Predpokladal som teda jeho úspešné nahratie do pamäte mikrokontroléru *SAM3X8E.* Po pripojení som cez externú linku už nevidel výpis generovaný predtým nahratým príkladom 3 ani nič ďalšie, pretože sériová komunikácia je v tomto príklade využitá len pri spustení systému. Funkčnosť nahratého programu som si teda overil pripojením LED diód k inicializovaným PWM portom. Až tento test ukázal funkčnosť a správnosť nahratého programu, pretože LED diódy postupne zhasínali a rozsvecovali podľa meniacej sa frekvencie PWM presne, ako program definoval.

```
C:\Users\Adam Zahatlan\Documents\Atmel Studio\7.0\PWM_PWM_LED_EXAMPLE1\
    PWM_PWM_LED_EXAMPLE1\Debug>mode COM8:1200,n,8,1

Status for device COM8:
-----------------------
    Baud:            1200
    Parity:          None

...

C:\Users\Adam Zahatlan\Documents\Atmel Studio\7.0\PWM_PWM_LED_EXAMPLE1\
    PWM_PWM_LED_EXAMPLE1\Debug>"C:\Program Files (x86)\Arduino\hardware\tools\
    bossac.exe" --port=COM8 -U false -e -w -v -b "C:\Users\Adam Zahatlan\
    Documents\Atmel Studio\7.0\PWM_PWM_LED_EXAMPLE1\PWM_PWM_LED_EXAMPLE1\Debug\
    PWM_PWM_LED_EXAMPLE1.bin" -R

...

chipId=0x285e0a60
Erase flash
Write 13488 bytes to flash

[                            ] 0% (0/53 pages)
[=====                       ] 18% (10/53 pages)
[===========                 ] 37% (20/53 pages)
[================            ] 56% (30/53 pages)
[======================      ] 75% (40/53 pages)
[============================ ] 94% (50/53 pages)
[=============================] 100% (53/53 pages)
Verify 13488 bytes of flash

...

[=============================] 100% (53/53 pages)
Verify successful
Set boot flash true
CPU reset.
```

Výpis 5: Sekvencia nahratia programu externým programátorom

4.4 Test generovania šírky pulzu pomocou PWM

K overeniu funkčnosti PWM som spočiatku používal LED. Po úspešnej inicializácií PWM portov, nastavení hodín a šírky pulzu sa ale z LED už nedalo veľa vyčítať. Preto bol overeniu funkčnosti generovania správnych PWM pulzov použitý osciloskop. Postupne som si generoval rôzne šírky pulzov na rôznych PWM portoch. Ako je možno vidieť na obrázkoch 20, pulzy sa generovali korektne so skoro nulovou časovou stratou.



(a) Šírka pulzu 1.5 ms **(b)** Generovanie puzlov 20 ms

Obrázek 20: Ukážka generovania pulzu 1.5 ms (obr. 20a) v časovom intervale 20 ms (obr. 20b) na osciloskope

4.5 Test riadiaceho programu

Neodbytnou vecou po naprogramovaní riadiaceho programu je vyskúšanie jeho funkčnosti. Overením generovania správnej šírky pulzu som si bol istý, že zmena pulzov bude prebiehať korektne. Bolo však treba otestovať, či program správne prijíma posielaný protokol, overí si jeho správnosť, dokáže správne predať prijaté informácie k spracovaniu a v správnom čase meniť postupne šírku pulzu pre všetky modifikované kanály. Navrhol som program tak, aby všetky tieto informácie vypisoval priamo na sériovú linku.

Ako je vidieť vo výpise 6, po inicializácií spojenia a zadaní príkazu *#7* na sériovú linku program správne odpovedá správou *noInit*, čo znamená, že poloha servomotorov ešte nebola užívateľom upravená.

Po zadaní nesprávneho vstupu program správne zareaguje, upozorní používateľa a nepredá hodnoty ďalej k spracovaniu.

Po zadaní správneho vstupu program korektne celý vstup rozoberie. Funkcia `PwmTask` si následne rozdelí jednotlivé pulzy na kroky podľa toho, aké sú súčasne nastavené a za aký čas sa

majú nové pulzy nastaviť. Tieto pulzy začne postupne v jednotlivých časových krokoch nastavovať.

Zmenu šírky pulzov som si overil opätovným poslaním príkazu #7, ktorý nám korektne vrátil súčasne modifikované dĺžky pulzov. Môžem teda konštatovať, že UART komunikácia, spracovanie dát a ich nasledovné nastavenie funguje korektne.

Čo však nesedel bol čas, počas ktorého sa šírka pulzov menila. Príčinou však nebola chyba v programe, ale fakt, že na sériovú linku sa zapisovalo veľké množstvo dát pričom sériová linka nie je najrýchlejšia. Preto sa čas zmeny šírky pulzov predĺžil. Program som navrhol aj pre jednoduchú modifikáciu týchto výpisov. Riešením bolo odstránenie definície konštanty `DEBUG`. Po jej odstránení sa na sériovú linku neposielali jednotlivé výpisy. Nastavovanie šírky pulzu som sledoval na osciloskope, čím som si overil aj čas, za ktorý sa šírka pulzu zmenila. Po tomto teste sa teda dala považovať funkcia riadiaceho programu ako správna.

```
#7
noInit

#4 P800 #0 T30000
DEBUG(parse_position): Got new servo: 4
DEBUG(parse_position): Got new pulse: 800
DEBUG(parse_position): Got new servo: 0
DEBUG(parse_position): Malformed message
DEBUG(uart_console): Received invalid / revoked position

#0 P833 #1 P2300 #17 P2300 #2 P1500 #3 P500 #4 P2433 T4000
DEBUG(parse_position): Got new servo: 0
DEBUG(parse_position): Got new pulse: 833

...

DEBUG(parse_position): Got new servo: 4
DEBUG(parse_position): Got new pulse: 2433
DEBUG(parse_position): Got new TTF: 4000
DEBUG(parse_position): Parsing done
DEBUG(uart_console): Enqueuing new position [833, 2300, 1500, 500, 2433]

Recieved queue
DEBUG(PwmTask): Starting transition to a new position: [833, 2300, 1500, 500,
    2433], T=4000
DEBUG(PwmTask): Entering critical section
```

```
DEDEBUG(PwmTask): Leaving critical section
Iter: 1, 200, T=0
New intermediary position: [1278, 2365, 2416, 799, 1126]
DEBUG(PwmTask): Entering critical section
DEBUG(PwmTask): Leaving critical section

...

Iter: 199, 200, T=0
New intermediary position: [836, 2301, 1505, 502, 2426]
DEBUG(PwmTask): Entering critical section
DEBUG(PwmTask): Leaving critical section
Iter: 200, 200, T=0
New intermediary position: [833, 2300, 1500, 500, 2433]
DEBUG(PwmTask): Final position reached
DEBUG(PwmTask): Entering critical section
DEBUG(PwmTask): Leaving critical section

#7
0 P833 1 P2300 2 P1500 3 P500 4 P2433 5 P0 T4000
```

Výpis 6: Ukážka komunikácie po UART linke

4.6 Test generovania dráhy

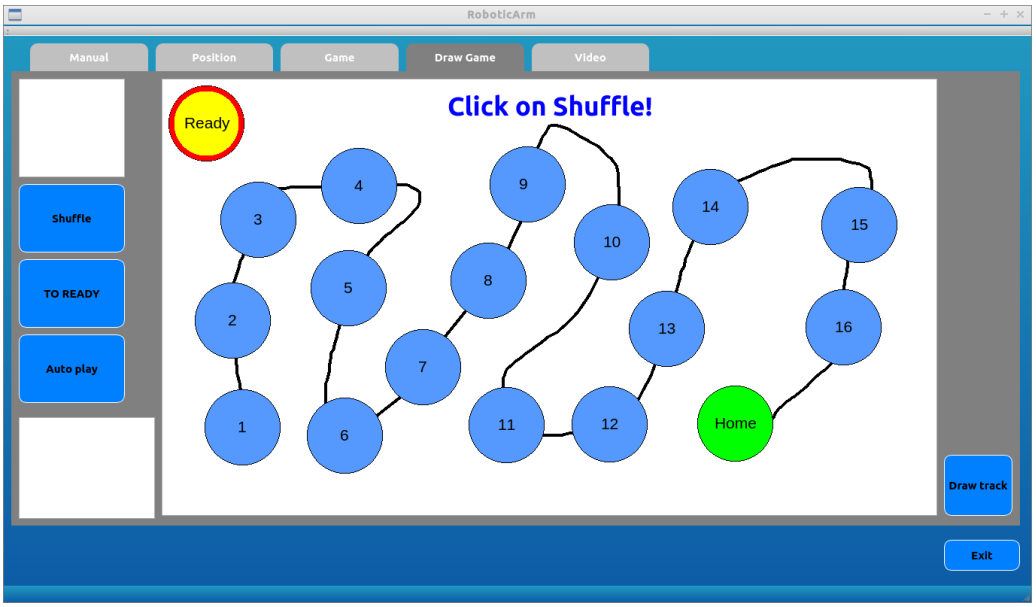
Testy kreslenia dráhy boli jednoduché. Pri jej kreslení som si zvolil počiatočný *Ready* bod v ľavom hornom kraji obrazovky. Užívateľ nemohol začať alebo skončiť s kreslením dráhy blízko pri tomto bode. Súčasne nemohol zájsť s kreslením ďalej ako je polomer kresleného bodu aby sa nevykreslil mimo obrazovky.

Kontrolovanie vzdialenosti generovaných bodov sa kontroluje už pri ťahaní prstom po obrazovke, a pokiaľ je vzdialenosť väčšia ako definovaná hodnota, pridajú sa súradnice bodu do poľa `positionPoints[]`, z ktorého sa vykresľujú pomocou triedy *QPainter*.

Dráha sa generovala a ukladala korektne. Problém robili mnou implementované chybové dialógy, ktoré sa pri skúške na dotykovej ploche nedarilo niekedy na prvý pokus stlačiť vinou menšej nepresnosti dotykového LCD displeja. Preto som tieto hlášky nakoniec odstránil. Vygenerovanú a načítanú dráhu je možno vidieť na obr. 21.

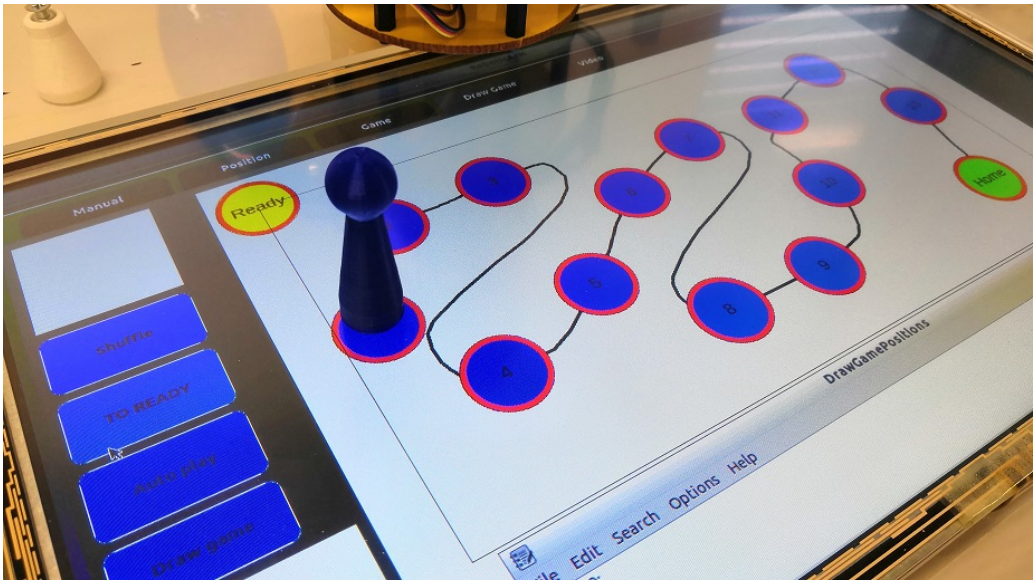
4.7 Test vypočítaných hodnôt pre umiestňovanie figúrky na dráhe

Ako ukázal test týchto vypočítaných hodnôt, figúrka sa v prvej polovici dráhy umiestňuje skoro presne na stred bodu ako je vidieť aj na obr. 22. V druhej polovici sú malé odchýlky, ktoré sa dajú minimalizovať kalibráciami v programe. Ako autor práce mám podozrenie na nepresnosť výpočtov, kedy vstupné vzdialenosti nie sú presne zmerané a vzniká tak pri zväčšujúcich sa vzdialenostiach aj odchýlka vo výpočtoch. Tá je však zanedbateľná k veľkosti hracej plochy a faktu, že robotická ruka je osadená slabými servomotormi, ktoré taktiež nevynikajú veľkou presnosťou a silou. Preto v určitých polohách môže byť táto odchýlka väčšia ako v iných polohách. To však už nebola náplň mojej práce. Pre tú stačilo, že sa robotická ruka hýbala a umiestňovala figúrku s celkom dobrou presnosťou.

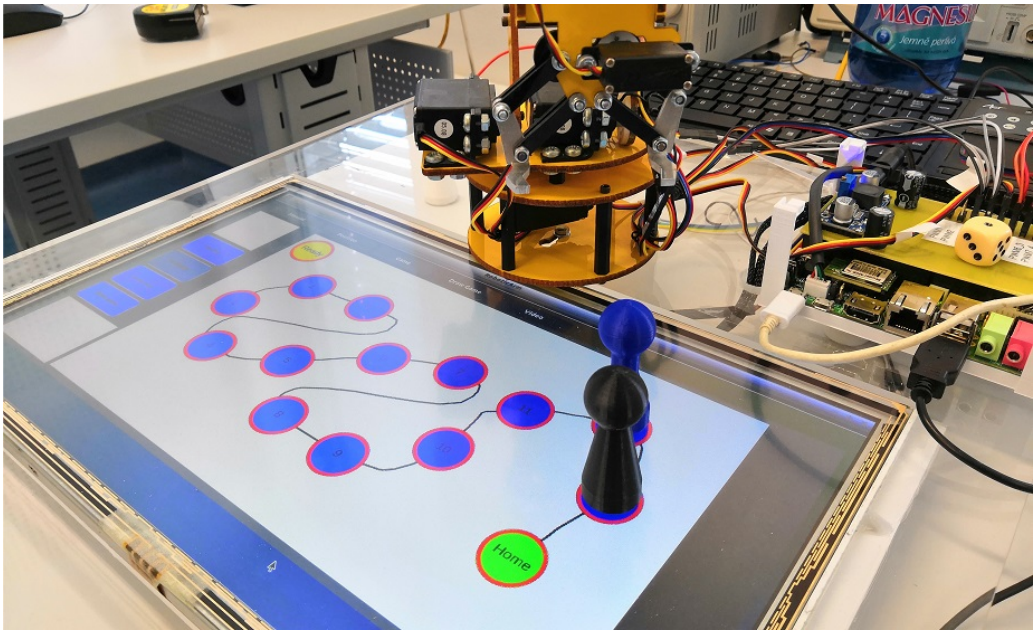


Obrázek 21: Dráha nakreslená užívateľom v aplikácií RoboticArm

Ďalej bolo na rade vyskúšanie funkčnosti hry dvoch hráčov, kedy mohol hrať užívateľ proti počítaču alebo druhému hráčovi. Tu som však narazilo na veľké obmedzenie, ktoré je spomenuté v kapitole 3.6.4. Robotická ruka je príliš nemotorná, jej klieštiny pre úchyt sú moc veľké a tak dochádzalo medzi figúrkami k ich vzájomnému zhadzovaniu v pozícií ako je vidieť na obr. 23. Po pokuse vyriešiť tento problém oddialením bodov sa síce kolízie umiernili, avšak kvalita dráhy značne klesla. Preto sa od tejto možnosti hry nakoniec odstúpilo.



Obrázek 22: Umiestnenie figúrky na kreslenej hracej ploche



Obrázek 23: Poloha figúrok, kedy dochádzalo k ich kolíziam

5 Závěr

Cieľom tejto práce bol návrh riadiaceho programu pre ovládanie robotickej ruky ako aj úprava užívateľskej aplikácie RoboticArm.

V prvej kapitole bola predstavená nová užívateľsky dostupná doska UDOO QUAD a vývojová doska Arduino DUE, ktorá je jej súčasťou. Obe tieto platformy boli bližšie popísané. Taktiež bol opísaný riadiaci program, ktorý bol vytvorený pre riadenie pohybov robotickej ruky. Program sa stará o správne nastavenie PWM výstupov a taktiež obsluhuje ich následnú zmenu parametrov podľa vstupu, ktorý pošle užívateľ. Preto si musí skontrolovať správnosť tohto vstupu počas toho, ako si ho delí na jednotlivé časti. Pokiaľ by prišiel nesprávny vstup, nesmie program prijaté dáta predať k ďalšiemu spracovaniu, inak by došlo k nepredvídateľnému chovaniu. Taktiež musí kontrolovať, či neprišla požiadavka na virtuálny kanál *#7*, kedy užívateľ nepožaduje nastavenie PWM výstupov, ale žiada ich súčasnú polohu. Pre jednoduchšie riadenie týchto procesov je program založený na real-time operačnom systéme FreeRTOS bežiacom na mikrokontroléru *SAM3X8E,* ktorý obsluhuje jednotlivé úlohy ako je spracovanie vstupu a následne správne nastavenie PWM výstupov. Ďalej boli opísané zmeny nad aplikáciou RoboticArm. Ako prvé bolo potrebné túto aplikáciu preniesť na novú platformu UDOO QUAD, čo nakoniec nebol žiaden problém po úspešnej inštalácií programu Qt Creator a potrebných súčastí ako sú knižnice *OpenCV, GStreamer* alebo kompilátor *gcc.* Zmeny nad samotnou aplikáciou pozostávali hlavne nad implementáciou novej hracej plochy dynamicky kreslenej užívateľom počas behu programu. K tomu bola použitá trieda *QPainter* pre jednoduché maľovanie geometrických útvarov, ktorá je vytvorená práve pre tieto účely. Taktiež bol užívateľ zoznámený s upravenou linuxovou distribúciou UDOObuntu, ktorá sa stará o správny beh aplikácie RoboticArm na platforme *ARM* procesoru *i.MX6Quad.* Všetky tieto zmeny boli otestované a opísané v samostatnej kapitole.

Žiaľ sa pri testoch narazilo na obmedzenie zo strany robotickej ruky, kedy pri hre dvoch hráčov dochádzalo k vzájomnému zhadzovaniu figúrok oboch hráčov. Príčinou bol spôsob uchytenia predmetov, pre ktorý sú použité úchopné kliešte a taktiež celková nemotornosť robotickej ruky. Preto sa od tejto verzie hry nakoniec po dohode s vedúcim práce ustúpilo.

Toto riešenie sa môže zdať finálne, no stále sa nájdu ďalšie možnosti a úpravy. Ako prvé by určite stálo za zváženie prerobenie úchytu, ktorý je súčasťou robotickej ruky. Toto riešenie má obmedzenia, kvôli ktorým nemohli byť splnené všetky ciele tejto práce, ako je spomenuté v predchádzajúcom odseku a taktiež v kapitolách 3.6.4 a 4.7.

Ďalej môže byť upravená samotná robotická ruka, ktorá disponuje slabými servomotormi s veľkou vôľou a nepresnosťou, kvôli ktorým dochádza v určitých prípadoch k nesprávnemu umiestňovaniu figúrky. Tie by sa mohli vymeniť za silnejšie, kedy by došlo v zvýšeniu presnosti a nemuseli by byť použité spriahnuté dva servomotory pre rameno, čo by taktiež ušetrilo jeden PWM port. Mohli by sa tak predĺžiť samotné ramená a zvýšiť celkový dosah a obratnosť ruky.

Ďalšie úpravy by sa mohli týkať LCD panelu, ktorý nevyniká veľkou presnosťou dotyku, kedy by detekoval položenú figúrku a nemuseli by tak jej súradnice byť uložené v programe.

Literatura

[1] PRCHAL, Aleš. *Řízení robotické ruky pomocí vícejádrového hybridního procesoru* [online]. Ostrava, 2017 [cit. 2019-04-23]. Dostupné z: https://theses.cz/id/1zjvq0/. Diplomová práce. Vysoká škola báňská - Technická univerzita Ostrava, Fakulta elektrotechniky a informatiky. Vedoucí práce Petr Olivka.

[2] SSC-32 Ver 2.0, Manual written for firmware version SSC32-1.06XE [online], c2005 [cit. 2019-04-17]. URL https://www.swarthmore.edu/NatSci/ceverba1/Class/e5/E5Lab2/ssc-32%20lynxmotion%20manual.pdf

[3] Qt APIs & Libraries, Tools and IDE [online], c2019 [cit. 2019-04-17]. URL https://www.qt.io/qt-features-libraries-apis-tools-and-ide/

[4] UDOO DUAL/QUAD board introduction [online], posledná revízia 16.1.2019 [cit. 2019-04-17]. URL https://www.udoo.org/docs/Introduction/Introduction.html

[5] UDOO WiKi [online], posledná revízia 30.6.2016 [cit. 2019-04-17]. URL https://elinux.org/UDOO

[6] ARM WiKi [online], posledná revízia 26.3.2019 [cit. 2019-04-17]. URL https://cs.wikipedia.org/wiki/ARM

[7] i.MX 6Dual/6Quad Applications Processors for Consumer Products [online], c2012-2018 [cit. 2019-04-17]. URL https://www.nxp.com/docs/en/data-sheet/IMX6DQCEC.pdf

[8] SAM3X / SAM3A Series, Atmel | SMART ARM-based MCU [online], c2015 [cit. 2019-04-17]. URL http://ww1.microchip.com/downloads/en/devicedoc/atmel-11057-32-bit-cortex-m3-microcontroller-sam3x-sam3a_datasheet.pdf

[9] UDOO, UDOObuntu [online], posledná revízia 16.1.2019 [cit. 2019-04-17]. URL https://www.udoo.org/docs/Software_&_OS_Distro/UDOObuntu.html

[10] UDOOBUNTU: THE OFFICIAL UDOO LINUX OPERATING SYSTEM [online], c2019 [cit. 2019-04-17]. URL https://www.udoo.org/udoobuntu-the-official-udoo-linux-operating-system/

[11] UDOO, i.MX6Q OPEN SOURCE MODULE, REV.D [online], c2013 [cit. 2019-04-17]. URL http://download.udoo.org/files/schematics/UDOO_REV_D_schematics.pdf

[12] CBT3125, Quadruple FET bus switch [online], c2018 [cit. 2019-04-17]. URL https://assets.nexperia.com/documents/data-sheet/CBT3125.pdf

[13] Microchip, Atmel Studio 7 [online], c1998-2019 [cit. 2019-04-17]. URL
https://www.microchip.com/mplab/avr-support/atmel-studio-7

[14] The FreeRTOS™ Kernel, Market Leading, De-facto Standard and Cross Platform RTOS kernel [online], [cit. 2019-04-17]. URL
https://www.freertos.org/

[15] Arduino forum, Topic: DUE shield Eagle file [online], c2019 [cit. 2019-04-17]. URL
https://forum.arduino.cc/index.php?topic=153875.0

A Konfigurácia Atmel Studia, pridanie vlastného programátora

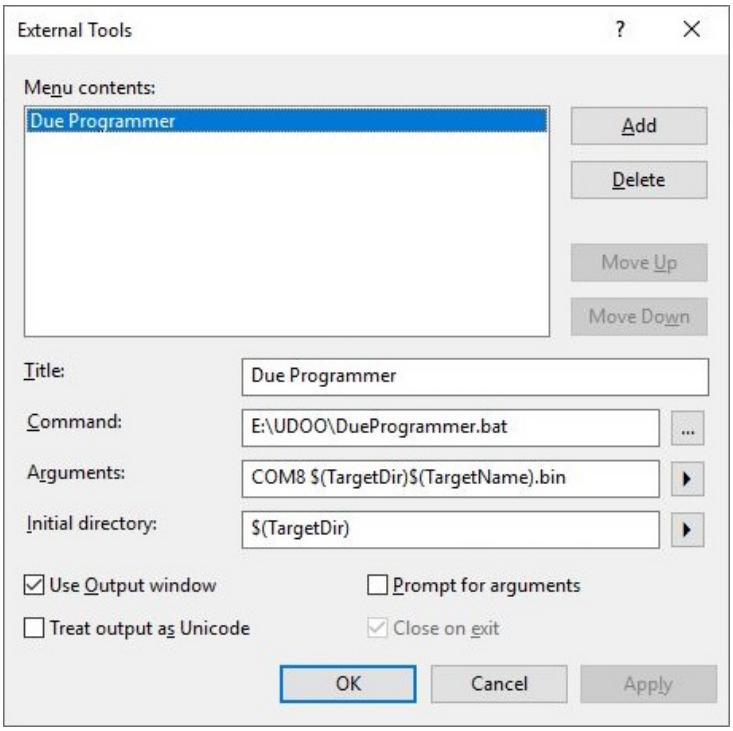
Pokiaľ chce užívateľ programovať Arduino DUE z externého PC, musí pridať „patch“ pre Arduino IDE dostupného na stránkach UDOO a modifikovať Atmel Studio:

1. Nainštalovať Arduino IDE
2. Stiahnuť potrebné súbory zo stránky `https://www.udoo.org/docs/Hackaton_Survival_Guide/Programming_UDOOs_Arduino_from_your_PC.html`
3. Nakopírovať stiahnuté súbory do inštalačnej lokácie Arduina IDE, napríklad *C:\Program Files (x86)\Arduino\hardware\tools*
4. Vytvoriť súbor *DueProgrammer.bat* a zapísať doň nasledujúce riadky z výpisu 7 (nezabudnúť zadať správnu cestu k stiahnutému *bossac.exe* súboru)
5. Otvoriť Atmel Studio, kliknúť na panel *Tools -> External Tools*
6. V *External Tools* okne kliknúť na *Add* a vyplniť nasledove, ako je vidieť aj na obr. 24
   - **Title -** Due Programmer
   - **Command -** cesta k *DueProgrammer.bat* súboru
   - **Arguments -** COM8 $(TargetDir)$(TargetName).bin (COM port nastaviť podľa toho, kde je pripojená doska)
   - **Initial directory -** $(TargetDir)
   - Kliknúť na *Apply* a *OK*
   - Zaškrtnúť checkbox *Use Output window*

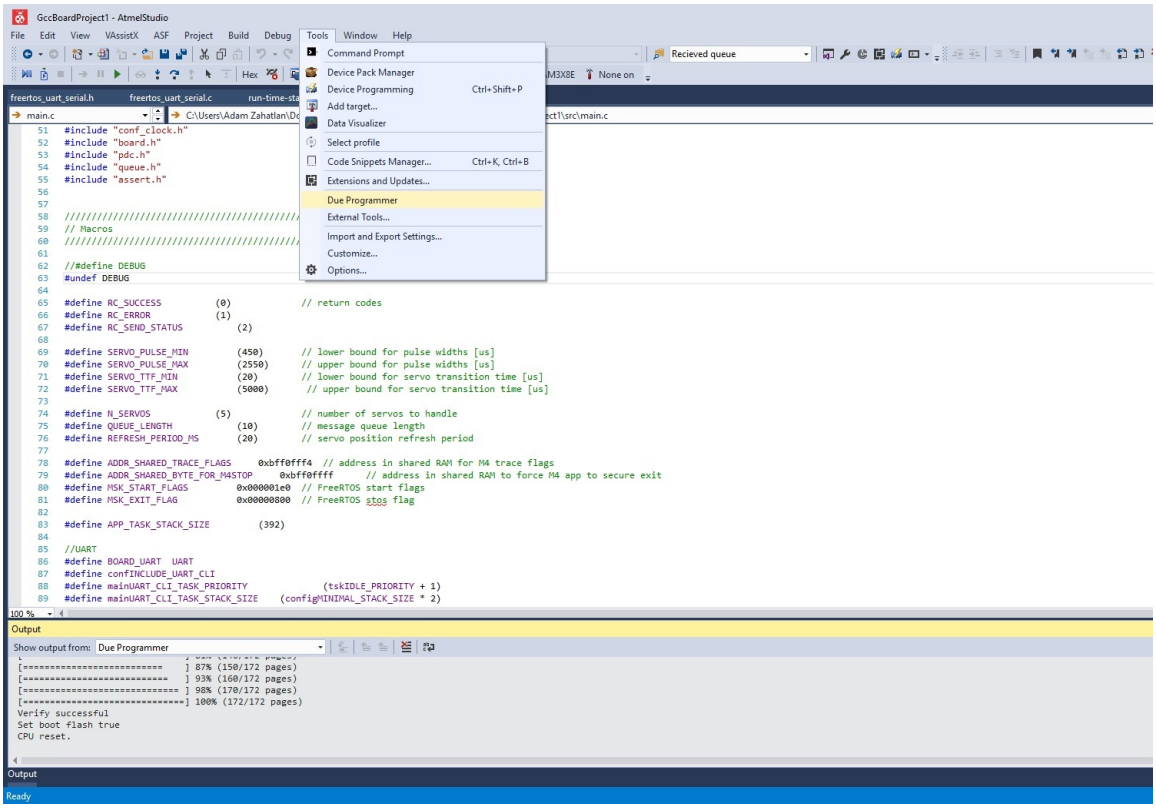
Takto sa užívateľovi v paneli *Tools* objaví programmer ukázaný na obr. 25, pomocou ktorého nahrá skompilovaný program do Arduina DUE.

```
mode %1:1200,n,8,1
sleep 1
"C:\Program Files (x86)\Arduino\hardware\tools\bossac.exe" --port=%1 -U false -
    e -w -v -b %2 -R
```

Výpis 7: Obsah súboru DueProgrammmer.bat



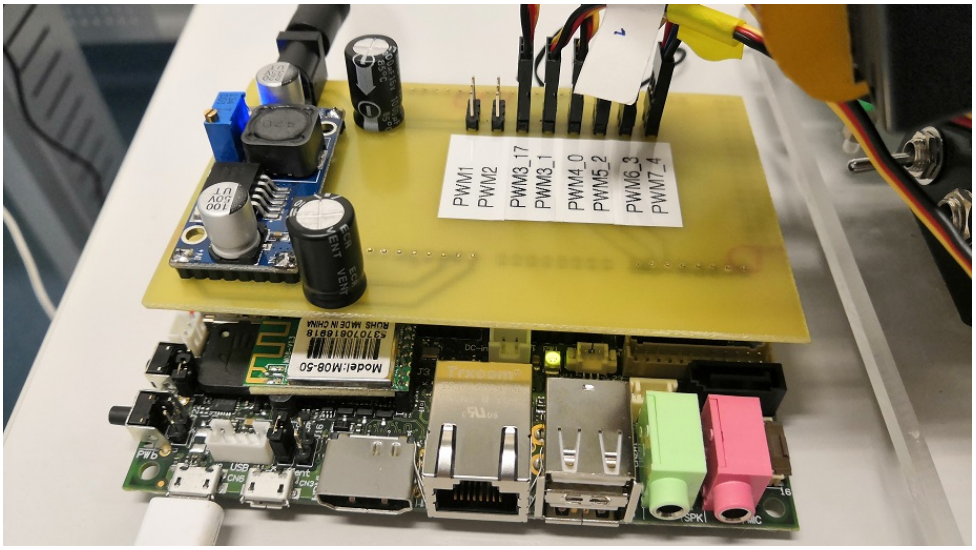
Obrázek 24: Nastavenie programmeru



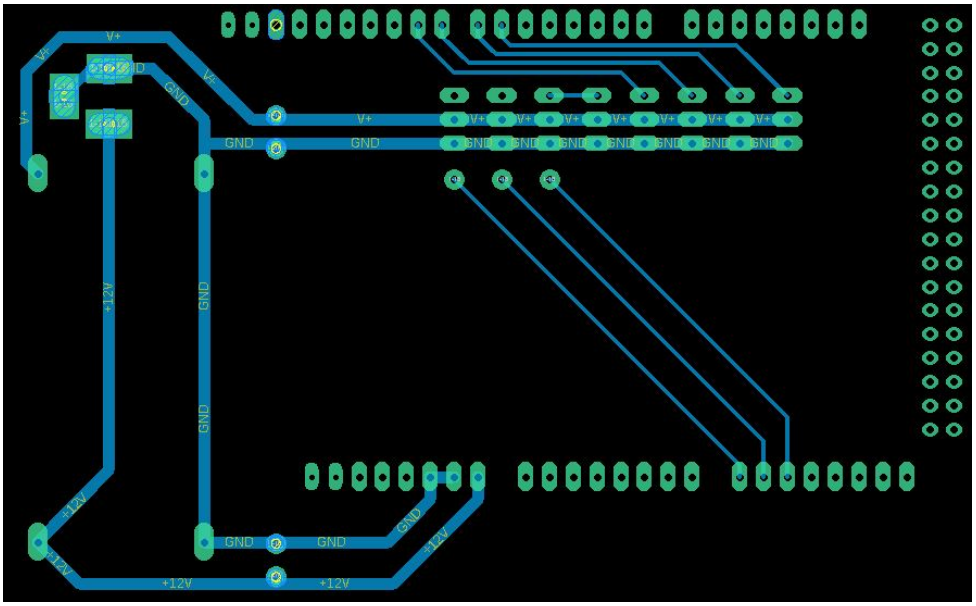
Obrázek 25: Modifikované Atmel Studio

B Rozširujúca doska pre Arduino DUE

Ako už bolo spomenuté v kapitole 3.3, bola mnou a vedúcim práce navrhnutá a vyrobená rozširujúca doska, ktorú je možno vidieť na obr. 26. Pre jej tvorbu bola použitá vytvorená knižnica [15] vytvorená Arduino komunitou a následne upravená mnou. V tejto prílohe môžeme taktiež vidieť DPS dosky na obr. 27, ktorý sa dá použiť pre vyrobenie ďalších klonov tohto shieldu.



Obrázek 26: Ukážka vyrobenej a zapojenej rozširujúcej dosky



Obrázek 27: Navrhnuté DPS rozširujúcej dosky

C Inštalácia QtCreator-u na architektúre ARM, nastavenie aplikácie RoboticArm

Ako som už spomínal v kapitole 3.5.2, desktopová aplikácia RoboticArm je navrhnutá a naprogramovaná vo frameworku *Qt*. Preto je pre jej spustenie mať tento framework nainštalovaný spoločne s prostredím *Qt Creator*, pomocou ktorého sa aplikácia vyvíjala a upravovala. V starším verziách *UDOObuntu 1* nebol *Qt framework* podporovaný balíkovacím systémom *apt* a musel sa inštalovať pomocou *gitu*. V novších verziách *UDOObuntu* je ho možno nainštalovať nasledovne:

1. Update balíčkov

   ```
   udooer@udoo:~$sudo apt-get update
   ```

2. Nainštalovanie Qt5 frameworku

   ```
   udooer@udoo:~$sudo apt-get install qt5-default
   ```

3. Nainštalovanie Qt Creator-u

   ```
   udooer@udoo:~$sudo apt-get install Qtcreator
   ```

4. Nainštalovanie potrebných súčastí

   ```
   udooer@udoo:~$sudo apt-get install cmake gcc libgstreamer1.0-0 \
   ↪  libgstreamer1.0-dev libopencv-dev
   ```

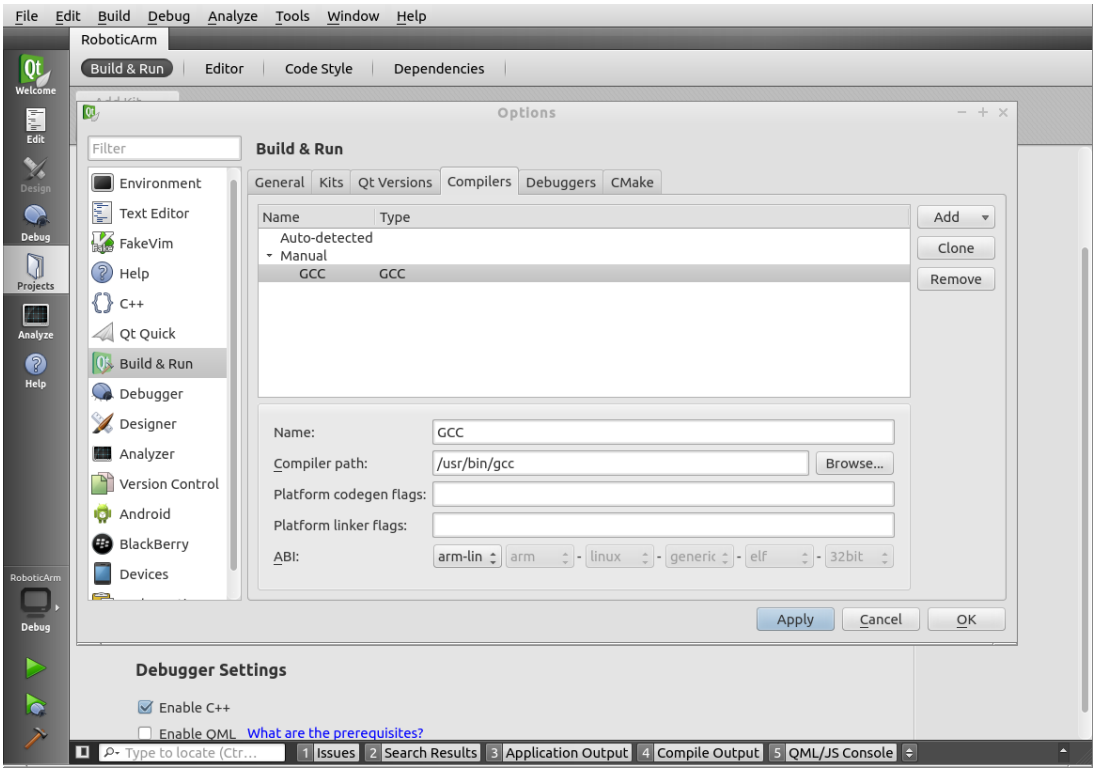
5. Stiahnutie a rozbalenie aplikácie RoboticArm

Teraz má užívateľ dve možnosti, ako spustiť aplikáciu RoboticArm.

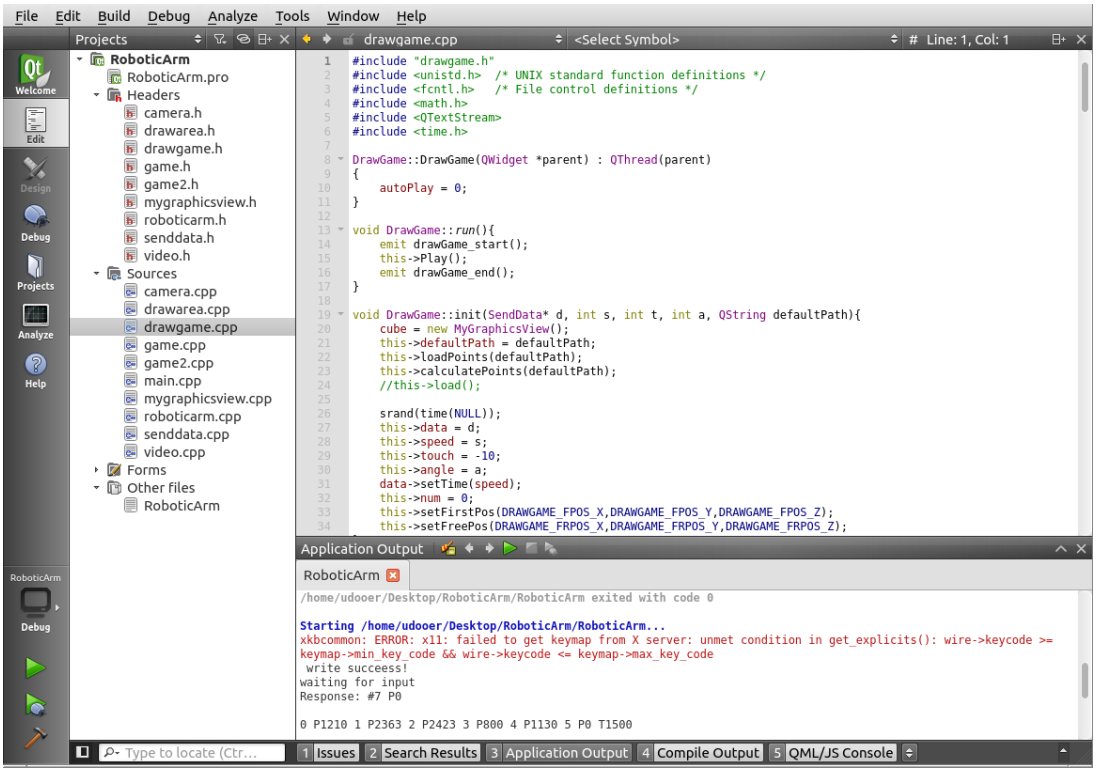
Prvá možnosť je prepnúť sa do priečinka rozbalenej aplikácie RoboticArm a spustiť nasledovné príkazy:

```
udooer@udoo:~$qmake
udooer@udoo:~$make
udooer@udoo:~$./RoboticArm
```

Druhá možnosť je spustiť Qt Creator, načítať súbor *RoboticArm.pro*, nastaviť cestu k *gcc* compileru, spustiť konfiguráciu projektu a spustiť aplikáciu. Na obrázku 28a je možno vidieť konfiguráciu *gcc* compileru a na obrázku 28b grafické rozhranie *Qt Creator*.



(a) Nastavenie gcc compileru



(b) Pripravený Qt Creator

Obrázek 28: Ukážka nastavenia programu Qt Creator

D Obsah priloženého média

- `/` –> Základná zložka prílohy
  - `documents/`
    - `CBT3125.pdf` –> Datasheet MOSFET tranzistora 2N7002
    - `IMX6DQCEC.pdf` –> Datasheet i.MX6Dual/Quad procesorov
    - `sam3x-sam3a_datasheet.pdf` –> Datasheet SAM3X/A mikrokontrolérov
    - `ssc-32 lynxmotion manual.pdf` –> Datasheet pre SSC-32 Servo Controller
    - `UDOO_REV_D_schematics.pdf` –> Schéma dosky UDOO DUAL/QUAD
    - `UDOO_Starting_Manual.PDF` –> Návod pre nového užívateľa dosky UDOO QUAD
  - `DUE Shield/`
    - `DUE_Shield_modified.lbr` –> Upravená knižnica rozširujúcej dosky pre UDOO QUAD
    - `DUE_Shield_original.lbr` –> Neupravená knižnica rozširujúcej dosky pre Arduino DUE
    - `Shield.brd` –> DPS rozširujúcej dosky pre UDOO QUAD
    - `Shield.sch` –> Schéma rozširujúcej dosky pre UDOO QUAD
  - `GccBoardProject1/`
    - `GccBoardProject1/` –> Zdrojové súbory riadiaceho programu
    - `GccBoardProject1.atsln` –> .atsln súbor pre otvorenie Atmel Studiom
  - `RoboticArm/` –> Zdrojové súbory aplikácie RoboticArm
  - `text/`
    - `Figures/` –> Zdrojové súbory a artefakty pre túto prácu
    - `2019_ZAH0068_BP.pdf` –> Elektronická verzia tejto práce
  - `Ukážka.mp4` –> Video ukazujúce funkčnosť tejto práce